Documentazione di LVGL v9.4md (Ita)

Release 9.4

LVGL Community

(Traduzione: Baldassarre Cesarano)

11 ago 2025

Indice

1 Introduzione 1

2 Per iniziare 7

3 Esempi 17

4 Integrazione e Driver 199

5 Common Widget Features 360

6 All Widgets 462

7 Main Modules 651

8 XML - Declarative UI 760

9 Moduli Ausiliari 799

10 3rd-Party Libraries 881

11 Debugging 914

12 Contribuire 922

13 Change Log 932

14 API 958

CAPITOLO 1

Introduzione

LVGL (Light and Versatile Graphics Library) è una libreria grafica C gratuita e open source che fornisce tutto il necessario per creare un'interfaccia utente grafica embedded con elementi grafici facili da usare, splendidi effetti visivi e un ridotto utilizzo di memoria.

LVGL fornisce il "motore GUI" che gestisce tutte le interazioni dell'utente finale con l'applicazione, tra cui non solo la visualizzazione del contenuto del display, l'accettazione dell'input dell'utente e la sua conversione in eventi "catturabili" dall'applicazione, ma anche oltre 30 widget integrati, oltre 100 proprietà di stile, scrolling [scorrimento], animazioni, rendering software avanzato, supporto GPU integrato da MCU a OpenGL e molto altro. La combinazione di questi elementi consente di sviluppare interfacce utente in modo efficiente senza dover reinventare la ruota.

L'utilizzo di LVGL in un progetto firmware riduce efficacemente il lavoro dello sviluppatore della GUI alla sua definizione e alla fornitura di hook di eventi appropriati per svolgere il prezioso lavoro della GUI.

LVGL può utilizzare GUI definite al 100% nel codice C scritto dall'utente (per cui qualsiasi aggiornamento alla GUI richiede la modifica del codice C e il re-flashing del firmware), oppure definizioni di GUI fornite in file XML (per cui gli aggiornamenti alla GUI richiedono solo la modifica dei file XML che definiscono la GUI per quel firmware), o qualsiasi combinazione di queste due opzioni adatta al progetto.

1.1 Caratteristiche Principali

- Elementi costitutivi potenti come *pulsanti, grafici, elenchi, slider, immagini*, ecc.
- Grafica avanzata con animazioni, anti-aliasing, opacità, scorrimento fluidog
- Vari dispositivi di input come touchpad, mouse, tastiera, encoder, ecc.
- Supporto multilingua con codifica UTF-8
- Supporto multi-display, anche con formati di colore misti
- Elementi grafici completamente personalizzabili con stili simili a CSS
- Indipendente dall'hardware: utilizzabile con qualsiasi microcontrollore o display
- Scalabile: in grado di funzionare con poca memoria (64 kB Flash, 16 kB RAM)
- *OS*, memoria esterna e *GPU* sono supportati ma non obbligatori
- Funzionamento con buffer di frame singolo anche con effetti grafici avanzati
- Scritto in C per la massima compatibilità (compatibile con C++)
- *Simulatore* per avviare la progettazione di GUI embedded su un PC senza hardware integrato

- Il codice utente sviluppato tramite simulatore può essere condiviso con il firmware per rendere più efficiente lo sviluppo dell'interfaccia utente.
- Binding a *MicroPython*
- Tutorial, esempi, temi per la progettazione rapida di interfacce grafiche (GUI)
- Documentazione disponibile online
- Gratuito e open source con licenza MIT
- Gratuito per progetti commerciali

1.1.1 Requisiti

In pratica, qualsiasi controller moderno in grado di pilotare un display è adatto all'esecuzione di LVGL. I requisiti minimi sono:

- Microcontrollore o processore a 16, 32 o 64 bit
- Velocità di clock consigliata >16 MHz
- Flash/ROM: >64 kB per i componenti essenziali (>180 kB consigliati)
- RAM:
  - Utilizzo di RAM statica: ~2 kB a seconda delle funzionalità utilizzate e dei tipi di Widget
  - stack: > 2kB (> 8 kB consigliati)
  - Dati dinamici (heap): >2 KB (>48 kB consigliati se si utilizzano molti widget GUI). Impostato da `LV_MEM_SIZE` in `lv_conf.h`.
  - Buffer di visualizzazione: > *"Risoluzione orizzontale"* pixel (>10 X *"Risoluzione orizzontale"*)
  - Un frame buffer nell'MCU o in un controller video esterno
- Compilatore C99 o successivo
- Conoscenza base del C (o C++):
  - puntatori.
  - struct.
  - callback.

> **Nota**
>
> *L'utilizzo della memoria può variare a seconda dell'architettura, del compilatore e delle opzioni di build.*

1.1.2 Licenza

Il progetto LVGL (inclusi tutti i repository) è concesso con la licenza MIT. Questo significa che lo si può utilizzare anche in progetti commerciali.

Non è obbligatorio, ma apprezzeremmo molto se si scrivesse qualche parola sul proprio progetto nella categoria My projects del forum o si inviasse un messaggio privato a lvgl.io.

Sebbene LVGL sia disponibile gratuitamente, dietro c'è un enorme lavoro. È stato creato da un gruppo di volontari che lo hanno reso disponibile a tutti nel loro tempo libero.

Per rendere sostenibile il progetto LVGL, preghiamo di considerare la possibilità di *Contribuire*. Si può scegliere tra diversi modi per contribuire. Consultare *Contribuire*, ad esempio scrivendo semplicemente un tweet su di sé usando LVGL, correggendo bug, traducendo la documentazione o addirittura diventando un manutentore.

1.1.3 FAQ

Dove posso porre domande?

Si possono porre domande nel forum: https://forum.lvgl.io/.

Utilizziamo GitHub issues per le discussioni relative allo sviluppo. Si dovrebbero utilizzarli solo se la domanda o il problema è strettamente correlato allo sviluppo della libreria.

Prima di postare una domanda, leggere questa sezione FAQ, poiché si potrebbe trovare la risposta al problema anche qui.

Questo MCU/hardware è supportato?

Tutti i MCU in grado di pilotare un display tramite porta parallela, SPI, interfaccia RGB o altro e che soddisfano i *Requisiti* sono supportati da LVGL.

Questo include:

- MCU "comuni" come STM32F, STM32H, NXP Kinetis, LPC, iMX, dsPIC33, PIC32, SWM341 ecc.
- Moduli Bluetooth, GSM e Wi-Fi come Nordic NRF, Espressif ESP32 e Raspberry Pi Pico W
- Linux con device frame buffer come /dev/fb0. Questo include computer Single-board come il Raspberry Pi
- Qualsiasi altro dispositivo con una MCU sufficientemente potente e una periferica per pilotare un display

Questo display è supportato?

LVGL richiede solo una semplice funzione driver per copiare un array di pixel in una determinata area del display. Se lo si può fare col proprio display, lo si può usare con LVGL.

Alcuni esempi dei tipi di display supportati:

- TFT con profondità di colore a 16 o 24 bit
- Monitor con porta HDMI
- Piccoli display monocromatici
- Display in scala di grigi
- anche matrici di LED
- o qualsiasi altro display in cui sia possibile controllare il colore/stato dei pixel

Consultare la sezione *Display (lv_display)* per ulteriori informazioni.

LVGL non si avvia, si blocca casualmente o non viene visualizzato nulla sul display. Quale potrebbe essere il problema?

- Provare ad aumentare `LV_MEM_SIZE`.
- Assicurarsi che il display funzioni senza LVGL. Ad esempio, coloralo in rosso all'avvio.
- Abilitare *Logging*.
- Abilitare le asserzioni in `lv_conf.h` (`LV_USE_ASSERT_...`).
- Se si usa un RTOS:
  - Aumentare la dimensione dello stack del task che chiama `lv_timer_handler()`.
  - Utilizzare uno dei metodi per la gestione dei thread descritti in *Considerazioni sui Thread*.

Il driver video non viene chiamato. Cos'è sfuggito?

Chiamare `lv_tick_inc(x)` come prescritto in *Tick Interface* e chiamare `lv_timer_handler()` come prescritto in *Gestione del Timer*.

Per saperne di più, consultare le sezioni *Tick Interface* e *Gestione del Timer*.

Perché il driver video viene chiamato una sola volta? Viene aggiornata solo la parte superiore del display.

Chiamare `lv_display_flush_ready(drv)` alla fine del proprio "*display flush callback*" come indicato nella sezione *Flush Callback*.

Perché si vede solo spazzatura sullo schermo?

Probabilmente c'è un bug nel driver del display. Provare il seguente codice senza usare LVGL. Si dovrebbe vedere un quadrato con una sfumatura rosso-blu.

```
#define BUF_WIDTH 255
uint16_t buf[BUF_WIDTH];
uint32_t i;
for(i = 0; i < BUF_WIDTH; i++) {
  lv_color_t c = lv_color_mix(lv_color_hex(0xff0000), lv_color_
↪hex(0x00ff00), i);
  buf[i] = lv_color_to_u16(c);

  lv_area_t a;
  a.x1 = 5;
  a.x2 = a.x1 + BUF_WIDTH - 1;
  a.y1 = 10 + i;
  a.y2 = 10 + i;
  my_flush_cb(NULL, &a, (void*) buf);
}
```

Perché si vedono colori senza senso sullo schermo?

Il formato colore LVGL configurato probabilmente non è compatibile con il formato colore del display. Controllare `LV_COLOR_DEPTH` in *lv_conf.h*.

Come si può velocizzare l'interfaccia utente?

- Attivare l'ottimizzazione del compilatore e abilitare la memorizzazione nella cache di istruzioni e dati, se disponibile sull'MCU.
- Aumentare la dimensione del buffer di visualizzazione.
- Utilizzare due buffer di visualizzazione e svuotare il buffer con DMA (o una periferica simile) in background.
- Aumentare la velocità di clock della porta SPI o parallela se si utilizzano per pilotare il display.
- Se il display ha una porta SPI, valutare la possibilità di passare a un modello con interfaccia parallela, poiché offre una velocità di elaborazione molto più elevata.
- Mantenere il buffer di visualizzazione nella RAM interna (non nella SRAM esterna) perché LVGL la utilizza molto e dovrebbe avere tempi di accesso rapidi.

Come posso ridurre l'utilizzo di flash/ROM?

Si possono disabilitare le funzionalità inutilizzate (come animazioni, file system, GPU, ecc.) e i tipi di widget in *lv_conf.h*.

Se si utilizza GCC/CLANG, è possibile aggiungere i flag di compilazione `-fdata-sections -ffunction-sections` e il flag del linker `--gc-sections` per rimuovere funzioni e variabili non utilizzate dal binario

finale. Se possibile, aggiungere il flag del compilatore `-flto` per abilitare il link-time-optimisation insieme a `-Os` per GCC o `-Oz` per CLANG e le versioni più recenti di GCC.

Come ridurre l'utilizzo di RAM?

- Ridurre la dimensione del *Display buffer*.
- Ridurre `LV_MEM_SIZE` in *lv_conf.h*. Questa memoria viene utilizzata quando si creano widget come pulsanti, etichette, ecc.
- Per lavorare con `LV_MEM_SIZE` inferiori, è possibile creare widget solo quando necessario ed eliminarli quando non sono più necessari.

Come si utilizza LVGL con un sistema operativo?

Per lavorare con un sistema operativo in cui le attività possono interrompersi a vicenda (in modo preventivo), è necessario assicurarsi che nessuna chiamata di funzione LVGL venga chiamata mentre è in corso un'altra chiamata LVGL. Esistono diversi modi per farlo. Consultare la sezione *Considerazioni sui Thread* per saperne di più.

1.1.4 Il Repository LVGL

Layout del repository

Tutti i repository del progetto LVGL sono ospitati su GitHub.

Lì si trovano i seguenti repository:

- lvgl: La libreria stessa con numerosi esempi e demo.
- blog: Sorgente del sito del blog
- sim: Sorgente del sito del simulatore online
- lv_port_*: Port LVGL per schede o ambienti di sviluppo
- lv_binding_*: Binding ad altri linguaggi

Policy di rilascio

I repository principali seguono le regole del Semantic Versioning:

- Versione major: modifiche API incompatibili. Es. v5.0.0, v6.0.0
- Versione minor: funzionalità nuove ma retrocompatibili. Es. v6.1.0, v6.2.0
- Versione patch: correzioni di bug retrocompatibili. Es. v6.1.1, v6.1.2

Tag come `vX.Y.Z` vengono creati per ogni release.

Ciclo di rilascio

- Correzioni di bug: rilasciate su richiesta, anche settimanalmente
- Release minor: ogni 3-4 mesi
- Release major: circa una volta all'anno

Branch

I repository principali hanno almeno i seguenti branch:

- `master`: versione più recente, le patch vengono unite direttamente qui
- `release/vX.Y`: versioni stabili delle release minor

Changelog

Le modifiche sono loggate in *Change Log*.

Versioni supportate

Prima della v8, l'ultima release minor di ogni serie principale era supportata per 1 anno. A partire dalla v8, ogni release minor è supportata per 1 anno.

Versione	Data di rilascio	Fine del supporto	Attivo
v5.3	1 Feb, 2019	1 Feb, 2020	No
v6.1	26 Nov, 2019	26 Nov, 2020	No
v7.11	16 Mar, 2021	16 Mar, 2022	No
v8.0	1 Jun, 2021	1 Jun, 2022	No
v8.1	10 Nov, 2021	10 Nov, 2022	No
v8.2	31 Gen, 2022	31 Gen, 2023	No
v8.3	6 Lug, 2022	1 Gen, 2025	No
v8.4	19 Mar, 2024	20 Mar, 2025	No
v9.0	22 Gen, 2024	22 Gen, 2025	No
v9.1	20 Mar, 2024	20 Mar, 2025	No
v9.2	26 Ago, 2024	26 Ago, 2025	Yes
v9.3	3 giugno 2025	3 giugno 2026	Yes

CAPITOLO 2

Per iniziare

2.1 Imparare le Basi

2.1.1 Panoramica

LVGL (Light and Versatile Graphics Library) è una libreria grafica gratuita e open source che fornisce tutto il necessario per creare un'interfaccia utente grafica embedded con elementi grafici facili da usare, simili a quelli dei telefoni cellulari, splendidi effetti visivi e un ridotto utilizzo della memoria.

LVGL può essere considerato come una raccolta di file C e H inseribili in qualsiasi progetto per aggiungere funzionalità di interfaccia utente al prodotto.

Con l'aiuto di funzioni API coerenti e facili da apprendere, è possibile creare widget (pulsanti, slider, grafici, ecc.), personalizzarli, aggiungere eventi, layout o animazioni.

In base a queste impostazioni, LVGL renderizzerà un'immagine (utilizzando il suo motore di rendering software nativo o una GPU) e chiamerà una funzione di callback per visualizzare l'immagine renderizzata sul display. Questa funzione di callback è l'interfaccia principale tra LVGL e il display. La maggior parte del lavoro relativo al porting si concentra sulla scrittura di tale callback in modo efficace.

Questo capitolo illustrerà le basi per dare un'idea del funzionamento di LVGL e di come può essere utilizzato. Per maggiori dettagli su ciascuna funzionalità, visitare la pagina di documentazione dedicata.

Data Flow di Base

1. **Inizializzazione del Driver**: È responsabilità dell'utente configurare l'orologio, i timer, le periferiche, ecc.
2. **Chiamata a lv_init()**: Inizializza LVGL stesso
3. **Creazione di dispositivi di visualizzazione e di input**: Si creano display (`lv_display_t`) e dispositivi di input (`lv_indev_t`) e si impostano i relativi callback
4. **Creazione dell'Interfaccia Utente**: Si chiamano le funzioni LVGL per creare schermate, widget, stili, animazioni, eventi, ecc.
5. **Chiamata a lv_timer_handler() in un loop**: Gestisce tutte le attività relative a LVGL:
   - aggiornare i(l) display,
   - leggere i dispositivi di input,
   - generare eventi in base all'input dell'utente (e altre cose),

- eseguire eventuali animazioni ed
- eseguire i timer creati dall'utente.

Esempio di integrazione

Questo è solo un breve esempio di come aggiungere LVGL a un nuovo progetto. Per maggiori dettagli, consultare *Aggiungere LVGL al Proprio Progetto*.

```
void main(void)
{
    your_driver_init();

    lv_init();

    lv_tick_set_cb(my_get_millis);

    lv_display_t * display = lv_display_create(320, 240);

    /*LVGL will render to this 1/10 screen sized buffer for 2 bytes/pixel*/
    static uint8_t buf[320 * 240 / 10 * 2];
    lv_display_set_buffers(display, buf, NULL, LV_DISPLAY_RENDER_MODE_
↪PARTIAL);

    /*This callback will display the rendered image*/
    lv_display_set_flush_cb(display, my_flush_cb);

    /*Create widgets*/
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Hello LVGL!");

    /*Make LVGL periodically execute its tasks*/
    while(1) {
        lv_timer_handler();
        my_sleep(5);  /*Wait 5 milliseconds before processing LVGL timer␣
↪again*/
    }
}

/*Return the elapsed milliseconds since startup.
 *It needs to be implemented by the user*/
uint32_t my_get_millis(void)
{
    return my_tick_ms;
}

/*Copy the rendered image to the screen.
 *It needs to be implemented by the user*/
void my_flush_cb(lv_display_t * disp, const lv_area_t * area, uint8_t * px_
↪buf)
{
    /*Show the rendered image on the display*/
    my_display_update(area, px_buf);

    /*Indicate that the buffer is available.
     *If DMA were used, call in the DMA complete interrupt*/
    lv_display_flush_ready();
}
```

2.1.2 Display

Display si riferisce all'hardware effettivo. Per collegare LVGL all'hardware, è necessario creare e inizializzare un oggetto `lv_display_t`.

LVGL supporta molti *driver integrati*, ma è facile anche inizializzare un display da zero (come mostrato sopra).

LVGL gestisce anche più display contemporaneamente.

2.1.3 Screen

Uno *Screen* è un widget LVGL creato su un *Display*. È un contenitore logico per altri widget. Un display può avere più screen, ma ce n'è sempre uno attivo, che può essere recuperato tramite `lv_screen_active()`. Restituisce un puntatore `lv_obj_t *`. Consultare *Lo Screen Attivo* per ulteriori informazioni.

Il modo più comune per creare uno screen è creare un *Widget di Base* con un genitore `NULL`. Ad esempio,

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

Uno screen può essere caricato in questo modo: `lv_screen_load(my_screen)`

2.1.4 I Widget

I widget sono gli elementi costitutivi fondamentali dell'interfaccia utente. Per esempio: *Button (lv_button)*, *Slider (lv_slider)*, *Drop-Down List (lv_dropdown)*, *Chart (lv_chart)*, ecc.

I widget possono essere creati dinamicamente chiamando le rispettive funzioni "create". La funzione *create* restituisce un puntatore `lv_obj_t *` utilizzabile per configurare il widget in un secondo momento.

Ogni funzione *create* ha un singolo argomento `parent` che definisce a quale widget verrà aggiunto quello nuovo.

Ad esempio:

```
lv_obj_t * my_button1 = lv_button_create(lv_screen_active());
lv_obj_t * my_label1 = lv_label_create(my_button1);
```

Se un widget o una schermata non sono più necessari, possono essere rimossi chiamando `lv_obj_delete(my_button1)`

Per modificare le proprietà dei widget, è possibile utilizzare due set di funzioni:

- Funzioni `lv_obj_...()` per proprietà comuni, ad esempio `lv_obj_set_width()`, `lv_obj_add_style()`, ecc. Queste sono trattate in *Common Widget Features*.
- Funzioni `lv_<widget_type>_...()` per proprietà specifiche del tipo, ad esempio `lv_label_set_text()`, `lv_slider_set_value()`, ecc.

Ecco un esempio che mostra anche alcune unità di misura *non-pixel* per le dimensioni:

```
lv_obj_t * my_button1 = lv_button_create(lv_screen_active());
/*Set parent-sized width, and content-sized height*/
lv_obj_set_size(my_button1, lv_pct(100), LV_SIZE_CONTENT);
/*Align to the right center with 20px offset horizontally*/
lv_obj_align(my_button1, LV_ALIGN_RIGHT_MID, -20, 0);

lv_obj_t * my_label1 = lv_label_create(my_button1);
lv_label_set_text_fmt(my_label1, "Click me!");
lv_obj_set_style_text_color(my_label1, lv_color_hex(0xff0000), 0);
/*Make the text red*/
```

Per visualizzare l'API completa di un widget, consultare la relativa documentazione in *All Widgets* o controllare il relativo file header nel codice sorgente.

2.1.5 Eventi

Gli eventi vengono utilizzati per informare l'applicazione che si è verificato un problema con un widget. È possibile assegnare una o più callback a un Widget, che verranno richiamate quando il Widget verrà cliccato, rilasciato, trascinato, eliminato, ecc.

Una callback viene assegnata in questo modo:

```
lv_obj_add_event_cb(btn, my_btn_event_cb, LV_EVENT_CLICKED, NULL);

...

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

È possibile utilizzare *LV_EVENT_ALL* al posto di *LV_EVENT_CLICKED* per richiamare la callback per tutti gli eventi.

Le callback degli eventi ricevono l'argomento `lv_event_t * e` contenente il codice dell'evento corrente e altre informazioni relative all'evento. Il codice dell'evento corrente può essere recuperato con:

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

Il Widget che ha attivato l'evento può essere recuperato con:

```
lv_obj_t * widget = lv_event_get_target_obj(e);
```

Si può scoprire tutto sugli eventi nella sezione *Eventi*.

2.1.6 Parti e Stati

Parti

I Widget sono costituiti da una o più *parti*. Ad esempio, un pulsante ha una sola *parte* chiamata *LV_PART_MAIN*. Tuttavia, uno *Slider (lv_slider)* ha *LV_PART_MAIN*, *LV_PART_INDICATOR* e *LV_PART_KNOB*.

Utilizzando le *parti* è possibile applicare diversi *stili* alle parti di un widget.

Consultare la documentazione del Widget per scoprire quali parti utilizza.

Stati

I Widget possono trovarsi in una combinazione dei seguenti stati:

- *LV_STATE_DEFAULT*: Normale, stato rilasciato
- *LV_STATE_CHECKED*: Stato attivato o selezionato
- *LV_STATE_FOCUSED*: Focus tramite tastiera o encoder o clic tramite touchpad/mouse
- *LV_STATE_FOCUS_KEY*: Focus tramite tastiera o encoder ma non tramite touchpad/mouse
- *LV_STATE_EDITED*: Modificato tramite encoder
- *LV_STATE_HOVERED*: Passaggio del mouse
- *LV_STATE_PRESSED*: Premuto
- *LV_STATE_SCROLLED*: In corso di scrolling
- *LV_STATE_DISABLED*: Disabilitato

Ad esempio, se si preme un Widget, questo passerà automaticamente agli stati *LV_STATE_FOCUSED* e *LV_STATE_PRESSED*. Quando lo si rilasca, lo stato *LV_STATE_PRESSED* verrà rimosso, mentre lo stato *LV_STATE_FOCUSED* rimarrà attivo.

Per verificare se un Widget si trova in un determinato stato, utilizzare `lv_obj_has_state(widget, LV_STATE_...)`. Restituirà `true` se il widget si trova attualmente in quello stato.

Per aggiungere o rimuovere stati a livello di codice, utilizzare:

```
lv_obj_add_state(widget, LV_STATE_...);
lv_obj_remove_state(widget, LV_STATE_...);
```

2.1.7 Stili

Inizializzazione degli stili

Gli stili sono contenuti in oggetti `lv_style_t`. Contengono proprietà come colore di sfondo, larghezza del bordo, font, ecc.

Gli stili possono essere aggiunti alla *Parte* e allo *Stato*. Solo il loro puntatore viene salvato nei Widget, quindi devono essere definiti come variabili statiche o globali.

Prima di utilizzare uno stile è necessario inizializzarlo con `lv_style_init(&style1)`. In seguito, è possibile aggiungere proprietà per configurare lo stile. Ad esempio:

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

Style Properties Overview per ulteriori dettagli.

Style Properties per vedere l'elenco completo.

Aggiungere stili ai widget

Successivamente è possibile aggiungere ai widget:

```
lv_obj_add_style(my_button1, &style1, 0); /*0 means add to the main part␣
↪and default state*/
lv_obj_add_style(my_checkbox1, &style1, LV_STATE_DISABLED); /*Add to␣
↪checkbox's disabled state*/
lv_obj_add_style(my_slider1, &style1, LV_PART_KNOB | LV_STATE_PRESSED); /
↪*Add to the slider's knob pressed state*/
```

Ereditarietà

Alcune proprietà (in particolare quelle relative al testo) possono essere ereditate. Ciò significa che se una proprietà non è impostata in un Widget, verrà cercata nei Widget genitore. Ad esempio, è possibile impostare il font una volta nello stile della schermata e tutto il testo su quella schermata lo ereditera per default, a meno che il font non sia specificato nel widget o in uno dei suoi genitori.

Stili Locali

È anche possibile aggiungere proprietà di stile locali ai Widget. Questo crea uno stile che risiede all'interno del Widget e viene utilizzato solo da quel Widget:

```
lv_obj_set_style_bg_color(slider1, lv_color_hex(0x2080bb), LV_PART_
↪INDICATOR | LV_STATE_PRESSED);
```

Stili per tutti i dettagli.

2.1.8 Soggetti e Osservatori

Soggetti e Osservatori sono strumenti potenti per creare facilmente data binding.

I soggetti sono variabili globali `lv_subject_t` che memorizzano valori interi, colori, stringhe, ecc.

L'interfaccia utente o l'applicazione possono sottoscrivere questi soggetti creando *callback dell'osservatore* che vengono notificati quando il soggetto cambia.

Anche un widget può sottoscrivere un soggetto. In questo modo, quando il widget viene eliminato, la sua sottoscrizione verrà automaticamente annullata.

Per alcuni widget, le funzioni di supporto semplificano la connessione ai soggetti. Ad esempio: `lv_slider_bind_value()`, `lv_label_bind_text()`.

In generale, l'utilizzo di *soggetti* e *osservatori* è un modo per collegare varie parti dell'interfaccia utente e farle reagire dinamicamente alle modifiche dei dati dell'applicazione, oppure consentire all'applicazione di reagire alle modifiche dell'interfaccia utente.

```
static void label_observer_cb(lv_observer_t * observer, lv_subject_t *␣
↪subject)
{
    lv_obj_t * label = lv_observer_get_target_obj(observer);
    lv_label_set_text_fmt(label, "Progress: %d", lv_subject_get_
↪int(subject));
}

...

static lv_subject_t subject1;
lv_subject_init_int(&subject1, 10);

lv_obj_t * label1 = lv_label_create(lv_screen_active());
/*lv_label_bind_text could have been used too*/
lv_subject_add_observer_obj(&subject1, label_observer_cb, label1, NULL);

lv_obj_t * slider1 = lv_slider_create(lv_screen_active());
lv_slider_bind_value(slider1, &subject1);
lv_obj_set_y(slider1, 30);

lv_subject_set_int(&subject1, 30);
```

Ulteriori approfondimenti nella pagina di documentazione degli *Osservatori*.

2.2 Esempi di Base

Di seguito sono riportati diversi esempi di base. Includono il codice applicativo che genera il Widget Tree necessario per far sì che LVGL esegua il rendering degli esempi mostrati. Ogni esempio presuppone che LVGL abbia subito una normale inizializzazione, ovvero che sia stato creato un oggetto `lv_display_t` e che quindi abbia un *Lo Screen Attivo*.

2.2.1 Un'etichetta *hello world* molto semplice

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_LABEL

/**
 * Basic example to create a "Hello world" label
 */
```

(continues on next page)

(continua dalla pagina precedente)

```
void lv_example_get_started_1(void)
{
    /*Change the active screen's background color*/
    lv_obj_set_style_bg_color(lv_screen_active(), lv_color_hex(0x003a57),␣
↪LV_PART_MAIN);

    /*Create a white label, set its text and align it to the center*/
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Hello world");
    lv_obj_set_style_text_color(lv_screen_active(), lv_color_hex(0xffffff),
↪ LV_PART_MAIN);
    lv_obj_align(label, LV_ALIGN_CENTER, 0, 0);
}

#endif
```

2.2.2 Un pulsante con un'etichetta e un evento di reazione al clic

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_BUTTON

static void btn_event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * btn = lv_event_get_target_obj(e);
    if(code == LV_EVENT_CLICKED) {
        static uint8_t cnt = 0;
        cnt++;

        /*Get the first child of the button which is the label and change␣
↪its text*/
        lv_obj_t * label = lv_obj_get_child(btn, 0);
        lv_label_set_text_fmt(label, "Button: %d", cnt);
    }
}

/**
 * Create a button with a label and react on click event.
 */
void lv_example_get_started_2(void)
{
    lv_obj_t * btn = lv_button_create(lv_screen_active());     /*Add a␣
↪button the current screen*/
    lv_obj_set_pos(btn, 10, 10);                               /*Set its␣
↪position*/
    lv_obj_set_size(btn, 120, 50);                             /*Set its␣
↪size*/
    lv_obj_add_event_cb(btn, btn_event_cb, LV_EVENT_ALL, NULL);           /
↪*Assign a callback to the button*/

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

(continues on next page)

(continua dalla pagina precedente)

```
}

#endif
```

2.2.3 Creare stili da zero per i pulsanti

```
#include "../lv_examples.h"
#if LV_USE_BUTTON && LV_BUILD_EXAMPLES

static lv_style_t style_btn;
static lv_style_t style_button_pressed;
static lv_style_t style_button_red;

static lv_color_t darken(const lv_color_filter_dsc_t * dsc, lv_color_t␣
↪color, lv_opa_t opa)
{
    LV_UNUSED(dsc);
    return lv_color_darken(color, opa);
}

static void style_init(void)
{
    /*Create a simple button style*/
    lv_style_init(&style_btn);
    lv_style_set_radius(&style_btn, 10);
    lv_style_set_bg_opa(&style_btn, LV_OPA_COVER);
    lv_style_set_bg_color(&style_btn, lv_palette_lighten(LV_PALETTE_GREY,␣
↪3));
    lv_style_set_bg_grad_color(&style_btn, lv_palette_main(LV_PALETTE_
↪GREY));
    lv_style_set_bg_grad_dir(&style_btn, LV_GRAD_DIR_VER);

    lv_style_set_border_color(&style_btn, lv_color_black());
    lv_style_set_border_opa(&style_btn, LV_OPA_20);
    lv_style_set_border_width(&style_btn, 2);

    lv_style_set_text_color(&style_btn, lv_color_black());

    /*Create a style for the pressed state.
     *Use a color filter to simply modify all colors in this state*/
    static lv_color_filter_dsc_t color_filter;
    lv_color_filter_dsc_init(&color_filter, darken);
    lv_style_init(&style_button_pressed);
    lv_style_set_color_filter_dsc(&style_button_pressed, &color_filter);
    lv_style_set_color_filter_opa(&style_button_pressed, LV_OPA_20);

    /*Create a red style. Change only some colors.*/
    lv_style_init(&style_button_red);
    lv_style_set_bg_color(&style_button_red, lv_palette_main(LV_PALETTE_
↪RED));
    lv_style_set_bg_grad_color(&style_button_red, lv_palette_lighten(LV_
↪PALETTE_RED, 3));
}

/**
```

(continues on next page)

(continua dalla pagina precedente)

```
 * Create styles from scratch for buttons.
 */
void lv_example_get_started_3(void)
{
    /*Initialize the style*/
    style_init();

    /*Create a button and use the new styles*/
    lv_obj_t * btn = lv_button_create(lv_screen_active());
    /* Remove the styles coming from the theme
     * Note that size and position are also stored as style properties
     * so lv_obj_remove_style_all will remove the set size and position␣
↪too */
    lv_obj_remove_style_all(btn);
    lv_obj_set_pos(btn, 10, 10);
    lv_obj_set_size(btn, 120, 50);
    lv_obj_add_style(btn, &style_btn, 0);
    lv_obj_add_style(btn, &style_button_pressed, LV_STATE_PRESSED);

    /*Add a label to the button*/
    lv_obj_t * label = lv_label_create(btn);
    lv_label_set_text(label, "Button");
    lv_obj_center(label);

    /*Create another button and use the red style too*/
    lv_obj_t * btn2 = lv_button_create(lv_screen_active());
    lv_obj_remove_style_all(btn2);                      /*Remove the␣
↪styles coming from the theme*/
    lv_obj_set_pos(btn2, 10, 80);
    lv_obj_set_size(btn2, 120, 50);
    lv_obj_add_style(btn2, &style_btn, 0);
    lv_obj_add_style(btn2, &style_button_red, 0);
    lv_obj_add_style(btn2, &style_button_pressed, LV_STATE_PRESSED);
    lv_obj_set_style_radius(btn2, LV_RADIUS_CIRCLE, 0); /*Add a local␣
↪style too*/

    label = lv_label_create(btn2);
    lv_label_set_text(label, "Button 2");
    lv_obj_center(label);
}

#endif
```

2.2.4 Creare uno slider e scriverne il valore su un'etichetta

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_SLIDER

static lv_obj_t * label;

static void slider_event_cb(lv_event_t * e)
{
    lv_obj_t * slider = lv_event_get_target_obj(e);

    /*Refresh the text*/
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_label_set_text_fmt(label, "%" LV_PRId32, lv_slider_get_
↪value(slider));
    lv_obj_align_to(label, slider, LV_ALIGN_OUT_TOP_MID, 0, -15);    /
↪*Align top of the slider*/
}

/**
 * Create a slider and write its value on a label.
 */
void lv_example_get_started_4(void)
{
    /*Create a slider in the center of the display*/
    lv_obj_t * slider = lv_slider_create(lv_screen_active());
    lv_obj_set_width(slider, 200);                          /*Set the␣
↪width*/
    lv_obj_center(slider);                                  /*Align to the␣
↪center of the parent (screen)*/
    lv_obj_add_event_cb(slider, slider_event_cb, LV_EVENT_VALUE_CHANGED,␣
↪NULL);     /*Assign an event function*/

    /*Create a label above the slider*/
    label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "0");
    lv_obj_align_to(label, slider, LV_ALIGN_OUT_TOP_MID, 0, -15);    /
↪*Align top of the slider*/
}

#endif
```

2.3 Prossimi Passi

Esistono diversi ottimi modi per approfondire la conoscenza di LVGL. Ecco un ordine consigliato di documenti da leggere e di spunti su cui esercitarsi nell'approfondire le proprie conoscenze:

1. Se non è stato già fatto, iniziare con la pagina *Introduzione* della documentazione. (5 minuti)
2. Consultate le Online Demos per vedere LVGL in azione. (3 minuti)
3. Se non è stato già fatto, leggere *Imparare le Basi* (sopra). (15 minuti)
4. Configura un *Simulatore su PC* LVGL. (10 minuti)
5. Consultare alcuni *Esempi* e il relativo codice.
6. *Aggiungere LVGL al progetto* o consultare i ready-to-use Projects.
7. Leggere le pagine di *Main Modules* per comprendere meglio la libreria. (2-3 ore)
8. Consultare la documentazione dei *All Widgets* per vedere cosa è disponibile.
9. Se ci sono domande, visitare il Forum.
10. Leggere la guida *Contribuire* per scoprire come contribuire a migliorare LVGL. (15 minuti)

CAPITOLO 3

Esempi

3.1 Per Iniziare

3.1.1 Un'etichetta *hello world* molto semplice

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_LABEL

/**
 * Basic example to create a "Hello world" label
 */
void lv_example_get_started_1(void)
{
    /*Change the active screen's background color*/
    lv_obj_set_style_bg_color(lv_screen_active(), lv_color_hex(0x003a57),␣
↪LV_PART_MAIN);

    /*Create a white label, set its text and align it to the center*/
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Hello world");
    lv_obj_set_style_text_color(lv_screen_active(), lv_color_hex(0xffffff),
↪ LV_PART_MAIN);
    lv_obj_align(label, LV_ALIGN_CENTER, 0, 0);
}

#endif
```

3.1.2 Un pulsante con un'etichetta e un evento di reazione al clic

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_BUTTON

static void btn_event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_t * btn = lv_event_get_target_obj(e);
    if(code == LV_EVENT_CLICKED) {
        static uint8_t cnt = 0;
        cnt++;

        /*Get the first child of the button which is the label and change␣
↪its text*/
        lv_obj_t * label = lv_obj_get_child(btn, 0);
        lv_label_set_text_fmt(label, "Button: %d", cnt);
    }
}

/**
 * Create a button with a label and react on click event.
 */
void lv_example_get_started_2(void)
{
    lv_obj_t * btn = lv_button_create(lv_screen_active());     /*Add a␣
↪button the current screen*/
    lv_obj_set_pos(btn, 10, 10);                            /*Set its␣
↪position*/
    lv_obj_set_size(btn, 120, 50);                          /*Set its␣
↪size*/
    lv_obj_add_event_cb(btn, btn_event_cb, LV_EVENT_ALL, NULL);           /
↪*Assign a callback to the button*/

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

#endif
```

3.1.3 Creare stili da zero per i pulsanti

```
#include "../lv_examples.h"
#if LV_USE_BUTTON && LV_BUILD_EXAMPLES

static lv_style_t style_btn;
static lv_style_t style_button_pressed;
static lv_style_t style_button_red;

static lv_color_t darken(const lv_color_filter_dsc_t * dsc, lv_color_t␣
↪color, lv_opa_t opa)
{
    LV_UNUSED(dsc);
    return lv_color_darken(color, opa);
}

static void style_init(void)
{
    /*Create a simple button style*/
    lv_style_init(&style_btn);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_set_radius(&style_btn, 10);
    lv_style_set_bg_opa(&style_btn, LV_OPA_COVER);
    lv_style_set_bg_color(&style_btn, lv_palette_lighten(LV_PALETTE_GREY,␣
↪3));
    lv_style_set_bg_grad_color(&style_btn, lv_palette_main(LV_PALETTE_
↪GREY));
    lv_style_set_bg_grad_dir(&style_btn, LV_GRAD_DIR_VER);

    lv_style_set_border_color(&style_btn, lv_color_black());
    lv_style_set_border_opa(&style_btn, LV_OPA_20);
    lv_style_set_border_width(&style_btn, 2);

    lv_style_set_text_color(&style_btn, lv_color_black());

    /*Create a style for the pressed state.
     *Use a color filter to simply modify all colors in this state*/
    static lv_color_filter_dsc_t color_filter;
    lv_color_filter_dsc_init(&color_filter, darken);
    lv_style_init(&style_button_pressed);
    lv_style_set_color_filter_dsc(&style_button_pressed, &color_filter);
    lv_style_set_color_filter_opa(&style_button_pressed, LV_OPA_20);

    /*Create a red style. Change only some colors.*/
    lv_style_init(&style_button_red);
    lv_style_set_bg_color(&style_button_red, lv_palette_main(LV_PALETTE_
↪RED));
    lv_style_set_bg_grad_color(&style_button_red, lv_palette_lighten(LV_
↪PALETTE_RED, 3));
}

/**
 * Create styles from scratch for buttons.
 */
void lv_example_get_started_3(void)
{
    /*Initialize the style*/
    style_init();

    /*Create a button and use the new styles*/
    lv_obj_t * btn = lv_button_create(lv_screen_active());
    /* Remove the styles coming from the theme
     * Note that size and position are also stored as style properties
     * so lv_obj_remove_style_all will remove the set size and position␣
↪too */
    lv_obj_remove_style_all(btn);
    lv_obj_set_pos(btn, 10, 10);
    lv_obj_set_size(btn, 120, 50);
    lv_obj_add_style(btn, &style_btn, 0);
    lv_obj_add_style(btn, &style_button_pressed, LV_STATE_PRESSED);

    /*Add a label to the button*/
    lv_obj_t * label = lv_label_create(btn);
    lv_label_set_text(label, "Button");
    lv_obj_center(label);

    /*Create another button and use the red style too*/
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_t * btn2 = lv_button_create(lv_screen_active());
    lv_obj_remove_style_all(btn2);                      /*Remove the␣
↪styles coming from the theme*/
    lv_obj_set_pos(btn2, 10, 80);
    lv_obj_set_size(btn2, 120, 50);
    lv_obj_add_style(btn2, &style_btn, 0);
    lv_obj_add_style(btn2, &style_button_red, 0);
    lv_obj_add_style(btn2, &style_button_pressed, LV_STATE_PRESSED);
    lv_obj_set_style_radius(btn2, LV_RADIUS_CIRCLE, 0); /*Add a local␣
↪style too*/

    label = lv_label_create(btn2);
    lv_label_set_text(label, "Button 2");
    lv_obj_center(label);
}

#endif
```

3.1.4 Creare uno slider e scriverne il valore su un'etichetta

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_SLIDER

static lv_obj_t * label;

static void slider_event_cb(lv_event_t * e)
{
    lv_obj_t * slider = lv_event_get_target_obj(e);

    /*Refresh the text*/
    lv_label_set_text_fmt(label, "%" LV_PRId32, lv_slider_get_
↪value(slider));
    lv_obj_align_to(label, slider, LV_ALIGN_OUT_TOP_MID, 0, -15);    /
↪*Align top of the slider*/
}

/**
 * Create a slider and write its value on a label.
 */
void lv_example_get_started_4(void)
{
    /*Create a slider in the center of the display*/
    lv_obj_t * slider = lv_slider_create(lv_screen_active());
    lv_obj_set_width(slider, 200);                          /*Set the␣
↪width*/
    lv_obj_center(slider);                                  /*Align to the␣
↪center of the parent (screen)*/
    lv_obj_add_event_cb(slider, slider_event_cb, LV_EVENT_VALUE_CHANGED,␣
↪NULL);     /*Assign an event function*/

    /*Create a label above the slider*/
    label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "0");
    lv_obj_align_to(label, slider, LV_ALIGN_OUT_TOP_MID, 0, -15);    /
↪*Align top of the slider*/
```

(continues on next page)

(continua dalla pagina precedente)

```
}

#endif
```

3.1.5 Semplice Gradiente Orizzontale

3.1.6 Gradiente (Obliquo) Lineare

3.1.7 Gradiente Radiale

3.1.8 Gradiente Conico

3.2 Stili

3.2.1 Stili Size

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_IMAGE

/**
 * Using the Size, Position and Padding style properties
 */
void lv_example_style_1(void)
{
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_radius(&style, 5);

    /*Make a gradient*/
    lv_style_set_width(&style, 150);
    lv_style_set_height(&style, LV_SIZE_CONTENT);

    lv_style_set_pad_ver(&style, 20);
    lv_style_set_pad_left(&style, 5);

    lv_style_set_x(&style, lv_pct(50));
    lv_style_set_y(&style, 80);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);

    lv_obj_t * label = lv_label_create(obj);
    lv_label_set_text(label, "Hello");
}

#endif
```

3.2.2 Stili Background

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

/**
 * Using the background style properties
```

(continues on next page)

(continua dalla pagina precedente)

```
 */
void lv_example_style_2(void)
{
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_radius(&style, 5);

    /*Make a gradient*/
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    static lv_grad_dsc_t grad;
    grad.dir = LV_GRAD_DIR_VER;
    grad.stops_count = 2;
    grad.stops[0].color = lv_palette_lighten(LV_PALETTE_GREY, 1);
    grad.stops[0].opa = LV_OPA_COVER;
    grad.stops[1].color = lv_palette_main(LV_PALETTE_BLUE);
    grad.stops[1].opa = LV_OPA_COVER;

    /*Shift the gradient to the bottom*/
    grad.stops[0].frac  = 128;
    grad.stops[1].frac  = 192;

    lv_style_set_bg_grad(&style, &grad);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_center(obj);
}

#endif
```

3.2.3 Stili Border

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

/**
 * Using the border style properties
 */
void lv_example_style_3(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    /*Set a background color and a radius*/
    lv_style_set_radius(&style, 10);
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_bg_color(&style, lv_palette_lighten(LV_PALETTE_GREY, 1));

    /*Add border to the bottom+right*/
    lv_style_set_border_color(&style, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_border_width(&style, 5);
    lv_style_set_border_opa(&style, LV_OPA_50);
    lv_style_set_border_side(&style, (lv_border_side_t)(LV_BORDER_SIDE_
↪BOTTOM | LV_BORDER_SIDE_RIGHT));
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_center(obj);
}

#endif
```

3.2.4 Stili Outline

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

/**
 * Using the outline style properties
 */
void lv_example_style_4(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    /*Set a background color and a radius*/
    lv_style_set_radius(&style, 5);
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_bg_color(&style, lv_palette_lighten(LV_PALETTE_GREY, 1));

    /*Add outline*/
    lv_style_set_outline_width(&style, 2);
    lv_style_set_outline_color(&style, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_outline_pad(&style, 8);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_center(obj);
}

#endif
```

3.2.5 Stili Shadow

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

/**
 * Using the Shadow style properties
 */
void lv_example_style_5(void)
{
    static lv_style_t style;
    lv_style_init(&style);

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

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_bg_color(&style, lv_palette_lighten(LV_PALETTE_GREY, 1));

    /*Add a shadow*/
    lv_style_set_shadow_width(&style, 55);
    lv_style_set_shadow_color(&style, lv_palette_main(LV_PALETTE_BLUE));

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_center(obj);
}

#endif
```

3.2.6 Stili Image

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_IMAGE

/**
 * Using the Image style properties
 */
void lv_example_style_6(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    /*Set a background color and a radius*/
    lv_style_set_radius(&style, 5);
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_bg_color(&style, lv_palette_lighten(LV_PALETTE_GREY, 3));
    lv_style_set_border_width(&style, 2);
    lv_style_set_border_color(&style, lv_palette_main(LV_PALETTE_BLUE));

    lv_style_set_image_recolor(&style, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_image_recolor_opa(&style, LV_OPA_50);
    lv_style_set_transform_rotation(&style, 300);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_image_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);

    LV_IMAGE_DECLARE(img_cogwheel_argb);
    lv_image_set_src(obj, &img_cogwheel_argb);

    lv_obj_center(obj);
}

#endif
```

3.2.7 Stili Arc

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

/**
 * Using the Arc style properties
 */
void lv_example_style_7(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    lv_style_set_arc_color(&style, lv_palette_main(LV_PALETTE_RED));
    lv_style_set_arc_width(&style, 4);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_arc_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_center(obj);
}
#endif
```

3.2.8 Stili Text

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_LABEL

/**
 * Using the text style properties
 */
void lv_example_style_8(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    lv_style_set_radius(&style, 5);
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_bg_color(&style, lv_palette_lighten(LV_PALETTE_GREY, 2));
    lv_style_set_border_width(&style, 2);
    lv_style_set_border_color(&style, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_pad_all(&style, 10);

    lv_style_set_text_color(&style, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_text_letter_space(&style, 5);
    lv_style_set_text_line_space(&style, 20);
    lv_style_set_text_decor(&style, LV_TEXT_DECOR_UNDERLINE);

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

    lv_obj_center(obj);
}
```

(continues on next page)

(continua dalla pagina precedente)

```
#endif
```

3.2.9 Stili Line

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_LINE

/**
 * Using the line style properties
 */
void lv_example_style_9(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    lv_style_set_line_color(&style, lv_palette_main(LV_PALETTE_GREY));
    lv_style_set_line_width(&style, 6);
    lv_style_set_line_rounded(&style, true);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_line_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);

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

    lv_obj_center(obj);
}

#endif
```

3.2.10 Transizione

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_IMAGE

/**
 * Creating a transition
 */
void lv_example_style_10(void)
{
    static const lv_style_prop_t props[] = {LV_STYLE_BG_COLOR, LV_STYLE_
↪BORDER_COLOR, LV_STYLE_BORDER_WIDTH, 0};

    /* A default transition
     * Make it fast (100ms) and start with some delay (200 ms)*/
    static lv_style_transition_dsc_t trans_def;
    lv_style_transition_dsc_init(&trans_def, props, lv_anim_path_linear,␣
↪100, 200, NULL);

    /* A special transition when going to pressed state
     * Make it slow (500 ms) but start  without delay*/
    static lv_style_transition_dsc_t trans_pr;
    lv_style_transition_dsc_init(&trans_pr, props, lv_anim_path_linear,␣
↪500, 0, NULL);
```

(continues on next page)

(continua dalla pagina precedente)

```
    static lv_style_t style_def;
    lv_style_init(&style_def);
    lv_style_set_transition(&style_def, &trans_def);

    static lv_style_t style_pr;
    lv_style_init(&style_pr);
    lv_style_set_bg_color(&style_pr, lv_palette_main(LV_PALETTE_RED));
    lv_style_set_border_width(&style_pr, 6);
    lv_style_set_border_color(&style_pr, lv_palette_darken(LV_PALETTE_RED,␣
↪3));
    lv_style_set_transition(&style_pr, &trans_pr);

    /*Create an object with the new style_pr*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style_def, 0);
    lv_obj_add_style(obj, &style_pr, LV_STATE_PRESSED);

    lv_obj_center(obj);
}

#endif
```

3.2.11 Utilizzo di stili multipli

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_IMAGE

/**
 * Using multiple styles
 */
void lv_example_style_11(void)
{
    /*A base style*/
    static lv_style_t style_base;
    lv_style_init(&style_base);
    lv_style_set_bg_color(&style_base, lv_palette_main(LV_PALETTE_LIGHT_
↪BLUE));
    lv_style_set_border_color(&style_base, lv_palette_darken(LV_PALETTE_
↪LIGHT_BLUE, 3));
    lv_style_set_border_width(&style_base, 2);
    lv_style_set_radius(&style_base, 10);
    lv_style_set_shadow_width(&style_base, 10);
    lv_style_set_shadow_offset_y(&style_base, 5);
    lv_style_set_shadow_opa(&style_base, LV_OPA_50);
    lv_style_set_text_color(&style_base, lv_color_white());
    lv_style_set_width(&style_base, 100);
    lv_style_set_height(&style_base, LV_SIZE_CONTENT);

    /*Set only the properties that should be different*/
    static lv_style_t style_warning;
    lv_style_init(&style_warning);
    lv_style_set_bg_color(&style_warning, lv_palette_main(LV_PALETTE_
↪YELLOW));
    lv_style_set_border_color(&style_warning, lv_palette_darken(LV_PALETTE_
```

(continues on next page)

(continua dalla pagina precedente)

```
↪YELLOW, 3));
    lv_style_set_text_color(&style_warning, lv_palette_darken(LV_PALETTE_
↪YELLOW, 4));

    /*Create an object with the base style only*/
    lv_obj_t * obj_base = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj_base, &style_base, 0);
    lv_obj_align(obj_base, LV_ALIGN_LEFT_MID, 20, 0);

    lv_obj_t * label = lv_label_create(obj_base);
    lv_label_set_text(label, "Base");
    lv_obj_center(label);

    /*Create another object with the base style and earnings style too*/
    lv_obj_t * obj_warning = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj_warning, &style_base, 0);
    lv_obj_add_style(obj_warning, &style_warning, 0);
    lv_obj_align(obj_warning, LV_ALIGN_RIGHT_MID, -20, 0);

    label = lv_label_create(obj_warning);
    lv_label_set_text(label, "Warning");
    lv_obj_center(label);
}

#endif
```

3.2.12 Stili Locali

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_IMAGE

/**
 * Local styles
 */
void lv_example_style_12(void)
{
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_bg_color(&style, lv_palette_main(LV_PALETTE_GREEN));
    lv_style_set_border_color(&style, lv_palette_lighten(LV_PALETTE_GREEN,␣
↪3));
    lv_style_set_border_width(&style, 3);

    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);

    /*Overwrite the background color locally*/
    lv_obj_set_style_bg_color(obj, lv_palette_main(LV_PALETTE_ORANGE), LV_
↪PART_MAIN);

    lv_obj_center(obj);
}

#endif
```

3.2.13 Aggiunta di stili a parti e a stati

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_IMAGE

/**
 * Add styles to parts and states
 */
void lv_example_style_13(void)
{
    static lv_style_t style_indic;
    lv_style_init(&style_indic);
    lv_style_set_bg_color(&style_indic, lv_palette_lighten(LV_PALETTE_RED,␣
↪3));
    lv_style_set_bg_grad_color(&style_indic, lv_palette_main(LV_PALETTE_
↪RED));
    lv_style_set_bg_grad_dir(&style_indic, LV_GRAD_DIR_HOR);

    static lv_style_t style_indic_pr;
    lv_style_init(&style_indic_pr);
    lv_style_set_shadow_color(&style_indic_pr, lv_palette_main(LV_PALETTE_
↪RED));
    lv_style_set_shadow_width(&style_indic_pr, 10);
    lv_style_set_shadow_spread(&style_indic_pr, 3);

    /*Create an object with the new style_pr*/
    lv_obj_t * obj = lv_slider_create(lv_screen_active());
    lv_obj_add_style(obj, &style_indic, LV_PART_INDICATOR);
    lv_obj_add_style(obj, &style_indic_pr, LV_PART_INDICATOR | LV_STATE_
↪PRESSED);
    lv_slider_set_value(obj, 70, LV_ANIM_OFF);
    lv_obj_center(obj);
}

#endif
```

3.2.14 Estensione del tema corrente

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_IMAGE
#include "../../lvgl_private.h"

static lv_style_t style_btn;

/*Will be called when the styles of the base theme are already added
  to add new styles*/
static void new_theme_apply_cb(lv_theme_t * th, lv_obj_t * obj)
{
    LV_UNUSED(th);

    if(lv_obj_check_type(obj, &lv_button_class)) {
        lv_obj_add_style(obj, &style_btn, 0);
    }
}

static void new_theme_init_and_set(void)
{
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Initialize the styles*/
    lv_style_init(&style_btn);
    lv_style_set_bg_color(&style_btn, lv_palette_main(LV_PALETTE_GREEN));
    lv_style_set_border_color(&style_btn, lv_palette_darken(LV_PALETTE_
↪GREEN, 3));
    lv_style_set_border_width(&style_btn, 3);

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

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

    /*Assign the new theme to the current display*/
    lv_display_set_theme(NULL, &th_new);
}

/**
 * Extending the current theme
 */
void lv_example_style_14(void)
{
    lv_obj_t * btn;
    lv_obj_t * label;

    btn = lv_button_create(lv_screen_active());
    lv_obj_align(btn, LV_ALIGN_TOP_MID, 0, 20);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Original theme");

    new_theme_init_and_set();

    btn = lv_button_create(lv_screen_active());
    lv_obj_align(btn, LV_ALIGN_BOTTOM_MID, 0, -20);

    label = lv_label_create(btn);
    lv_label_set_text(label, "New theme");
}

#endif
```

3.2.15 Opacità e trasformazioni

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_BUTTON && LV_USE_LABEL

/**
 * Opacity and Transformations
 */
void lv_example_style_15(void)
{
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_t * btn;
    lv_obj_t * label;

    /*Normal button*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_set_size(btn, 100, 40);
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, -70);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Normal");
    lv_obj_center(label);

    /*Set opacity
     *The button and the label is rendered to a layer first and that layer␣
↪is blended*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_set_size(btn, 100, 40);
    lv_obj_set_style_opa(btn, LV_OPA_50, 0);
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, 0);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Opa:50%");
    lv_obj_center(label);

    /*Set transformations
     *The button and the label is rendered to a layer first and that layer␣
↪is transformed*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_set_size(btn, 100, 40);
    lv_obj_set_style_transform_rotation(btn, 150, 0);        /*15 deg*/
    lv_obj_set_style_transform_scale(btn, 256 + 64, 0);   /*1.25x*/
    lv_obj_set_style_transform_pivot_x(btn, 50, 0);
    lv_obj_set_style_transform_pivot_y(btn, 20, 0);
    lv_obj_set_style_opa(btn, LV_OPA_50, 0);
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, 70);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Transf.");
    lv_obj_center(label);
}

#endif
```

3.2.16 Manopola metallica con gradiente conico

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

#if LV_USE_DRAW_SW_COMPLEX_GRADIENTS

/**
 * Simulate metallic knob using conical gradient
 * For best effect set LV_GRADIENT_MAX_STOPS to 8 or at least 3
 */
void lv_example_style_16(void)
```

(continues on next page)

(continua dalla pagina precedente)

```
{
#if LV_GRADIENT_MAX_STOPS >= 8
    static const lv_color_t grad_colors[8] = {
        LV_COLOR_MAKE(0xe8, 0xe8, 0xe8),
        LV_COLOR_MAKE(0xff, 0xff, 0xff),
        LV_COLOR_MAKE(0xfa, 0xfa, 0xfa),
        LV_COLOR_MAKE(0x79, 0x79, 0x79),
        LV_COLOR_MAKE(0x48, 0x48, 0x48),
        LV_COLOR_MAKE(0x4b, 0x4b, 0x4b),
        LV_COLOR_MAKE(0x70, 0x70, 0x70),
        LV_COLOR_MAKE(0xe8, 0xe8, 0xe8),
    };
#elif LV_GRADIENT_MAX_STOPS >= 3
    static const lv_color_t grad_colors[3] = {
        LV_COLOR_MAKE(0xe8, 0xe8, 0xe8),
        LV_COLOR_MAKE(0xff, 0xff, 0xff),
        LV_COLOR_MAKE(0x79, 0x79, 0x79),
    };
#else
    static const lv_color_t grad_colors[2] = {
        LV_COLOR_MAKE(0xe8, 0xe8, 0xe8),
        LV_COLOR_MAKE(0x79, 0x79, 0x79),
    };
#endif

    /*Create a style with gradient background and shadow*/
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_radius(&style, 500);
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_shadow_color(&style, lv_color_black());
    lv_style_set_shadow_width(&style, 50);
    lv_style_set_shadow_offset_x(&style, 20);
    lv_style_set_shadow_offset_y(&style, 20);
    lv_style_set_shadow_opa(&style, LV_OPA_50);

    /*First define a color gradient. In this example we use a gray color␣
↪map with random values.*/
    static lv_grad_dsc_t grad;

    lv_grad_init_stops(&grad, grad_colors, NULL, NULL, sizeof(grad_colors)␣
↪/ sizeof(lv_color_t));

    /*Make a conical gradient with the center in the middle of the object*/
#if LV_GRADIENT_MAX_STOPS >= 8
    lv_grad_conical_init(&grad, LV_GRAD_CENTER, LV_GRAD_CENTER, 0, 120, LV_
↪GRAD_EXTEND_REFLECT);
#elif LV_GRADIENT_MAX_STOPS >= 3
    lv_grad_conical_init(&grad, LV_GRAD_CENTER, LV_GRAD_CENTER, 45, 125,␣
↪LV_GRAD_EXTEND_REFLECT);
#else
    lv_grad_conical_init(&grad, LV_GRAD_CENTER, LV_GRAD_CENTER, 45, 110,␣
↪LV_GRAD_EXTEND_REFLECT);
#endif

    /*Set gradient as background*/
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_set_bg_grad(&style, &grad);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_set_size(obj, 200, 200);
    lv_obj_center(obj);
}

#else

void lv_example_style_16(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_set_width(label, LV_PCT(80));
    lv_label_set_text(label, "LV_USE_DRAW_SW_COMPLEX_GRADIENTS is not␣
↪enabled");
    lv_label_set_long_mode(label, LV_LABEL_LONG_MODE_SCROLL_CIRCULAR);
    lv_obj_center(label);
}

#endif /*LV_USE_DRAW_SW_COMPLEX_GRADIENTS*/

#endif /*LV_BUILD_EXAMPLES*/
```

3.2.17 Gradiente radiale come sfondo

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

#if LV_USE_DRAW_SW_COMPLEX_GRADIENTS

/**
 * Using radial gradient as background
 */
void lv_example_style_17(void)
{
    static const lv_color_t grad_colors[2] = {
        LV_COLOR_MAKE(0x9B, 0x18, 0x42),
        LV_COLOR_MAKE(0x00, 0x00, 0x00),
    };

    int32_t width = lv_display_get_horizontal_resolution(NULL);
    int32_t height = lv_display_get_vertical_resolution(NULL);

    static lv_style_t style;
    lv_style_init(&style);

    /*First define a color gradient. In this example we use a purple to␣
↪black color map.*/
    static lv_grad_dsc_t grad;

    lv_grad_init_stops(&grad, grad_colors, NULL, NULL, sizeof(grad_colors)␣
↪/ sizeof(lv_color_t));
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Make a radial gradient with the center in the middle of the object,␣
↪extending to the farthest corner*/
    lv_grad_radial_init(&grad, LV_GRAD_CENTER, LV_GRAD_CENTER, LV_GRAD_
↪RIGHT, LV_GRAD_BOTTOM, LV_GRAD_EXTEND_PAD);

    /*Set gradient as background*/
    lv_style_set_bg_grad(&style, &grad);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_set_size(obj, width, height);
    lv_obj_center(obj);
}

#else

void lv_example_style_17(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_set_width(label, LV_PCT(80));
    lv_label_set_text(label, "LV_USE_DRAW_SW_COMPLEX_GRADIENTS is not␣
↪enabled");
    lv_label_set_long_mode(label, LV_LABEL_LONG_MODE_SCROLL_CIRCULAR);
    lv_obj_center(label);
}

#endif /*LV_USE_DRAW_SW_COMPLEX_GRADIENTS*/

#endif /*LV_BUILD_EXAMPLES*/
```

3.2.18 Gradienti per lo sfondo del pulsante

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

#if LV_USE_DRAW_SW_COMPLEX_GRADIENTS

/**
 * Using various gradients for button background
 */
void lv_example_style_18(void)
{
    static const lv_color_t grad_colors[2] = {
        LV_COLOR_MAKE(0x26, 0xa0, 0xda),
        LV_COLOR_MAKE(0x31, 0x47, 0x55),
    };

    /*Create a linear gradient going from the top left corner to the␣
↪bottom at an angle, with reflected color map*/
    static lv_style_t style_with_linear_gradient_bg;
    static lv_grad_dsc_t linear_gradient_dsc;     /*NOTE: the gradient␣
↪descriptor must be static or global variable!*/

    lv_style_init(&style_with_linear_gradient_bg);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_grad_init_stops(&linear_gradient_dsc, grad_colors, NULL, NULL,␣
↪sizeof(grad_colors) / sizeof(lv_color_t));
    lv_grad_linear_init(&linear_gradient_dsc, lv_pct(0), lv_pct(0), lv_
↪pct(20), lv_pct(100), LV_GRAD_EXTEND_REFLECT);
    lv_style_set_bg_grad(&style_with_linear_gradient_bg, &linear_gradient_
↪dsc);
    lv_style_set_bg_opa(&style_with_linear_gradient_bg, LV_OPA_COVER);

    /*Create a radial gradient with the center in the top left 1/3rd of␣
↪the object, extending to the bottom right corner, with reflected color␣
↪map*/
    static lv_style_t style_with_radial_gradient_bg;
    static lv_grad_dsc_t radial_gradient_dsc;     /*NOTE: the gradient␣
↪descriptor must be static or global variable!*/

    lv_style_init(&style_with_radial_gradient_bg);
    lv_grad_init_stops(&radial_gradient_dsc, grad_colors, NULL, NULL,␣
↪sizeof(grad_colors) / sizeof(lv_color_t));
    lv_grad_radial_init(&radial_gradient_dsc, lv_pct(30), lv_pct(30), lv_
↪pct(100), lv_pct(100), LV_GRAD_EXTEND_REFLECT);
    lv_style_set_bg_grad(&style_with_radial_gradient_bg, &radial_gradient_
↪dsc);
    lv_style_set_bg_opa(&style_with_radial_gradient_bg, LV_OPA_COVER);

    /*Create buttons with different gradient styles*/

    lv_obj_t * btn;
    lv_obj_t * label;

    /*Simple horizontal gradient*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_set_style_bg_color(btn, grad_colors[0], 0);
    lv_obj_set_style_bg_grad_color(btn, grad_colors[1], 0);
    lv_obj_set_style_bg_grad_dir(btn, LV_GRAD_DIR_HOR, 0);
    lv_obj_set_size(btn, 150, 50);
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, -100);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Horizontal");
    lv_obj_center(label);

    /*Simple vertical gradient*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_set_style_bg_color(btn, grad_colors[0], 0);
    lv_obj_set_style_bg_grad_color(btn, grad_colors[1], 0);
    lv_obj_set_style_bg_grad_dir(btn, LV_GRAD_DIR_VER, 0);
    lv_obj_set_size(btn, 150, 50);
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, -40);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Vertical");
    lv_obj_center(label);

    /*Complex linear gradient*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_add_style(btn, &style_with_linear_gradient_bg, 0);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_size(btn, 150, 50);
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, 20);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Linear");
    lv_obj_center(label);

    /*Complex radial gradient*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_add_style(btn, &style_with_radial_gradient_bg, 0);
    lv_obj_set_size(btn, 150, 50);
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, 80);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Radial");
    lv_obj_center(label);
}

#else

void lv_example_style_18(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_set_width(label, LV_PCT(80));
    lv_label_set_text(label, "LV_USE_DRAW_SW_COMPLEX_GRADIENTS is not␣
↪enabled");
    lv_label_set_long_mode(label, LV_LABEL_LONG_MODE_SCROLL_CIRCULAR);
    lv_obj_center(label);
}

#endif /*LV_USE_DRAW_SW_COMPLEX_GRADIENTS*/

#endif /*LV_BUILD_EXAMPLES*/
```

3.2.19 Test tra stile di ricolorazione o sfondo modale pieno

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_SLIDER && LV_USE_LOG

/**
 * Test between a full background modal and a recolor modal
 */
void lv_example_style_19(void)
{
    /*Add lv_example_style_11 as background*/
    lv_example_style_11();

    /* Set to 1 to enable recolor overlay instead of solid background */
#if 0
    /* Apply a screen-wide tint using recolor (efficient overlay).
     * This modifies the visual appearance by blending a semi-transparent␣
↪color
     * over existing content without creating additional objects.
     * It's lighter on performance compared to a full-size background␣
↪object. */
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_style_recolor(lv_screen_active(), lv_color_black(), 0);
    lv_obj_set_style_recolor_opa(lv_screen_active(), LV_OPA_50, 0);
#else
    /* Simulate a modal background by setting a semi-transparent black␣
↪background
     * on lv_layer_top(), the highest built-in layer.
     * This method creates a new full-screen object and can consume more␣
↪resources
     * compared to recolor, especially when using images or gradients. */
    lv_obj_set_style_bg_color(lv_layer_top(), lv_color_black(), 0);
    lv_obj_set_style_bg_opa(lv_layer_top(), LV_OPA_50, 0);
#endif

    lv_obj_t * obj = lv_slider_create(lv_layer_top());
    lv_obj_center(obj);

    lv_refr_now(NULL); /*Update layouts and render*/

    lv_obj_invalidate(lv_screen_active());

    uint32_t t = lv_tick_get();
    lv_refr_now(NULL); /*Render only*/
    LV_LOG_USER("%" LV_PRIu32 " ms\n", lv_tick_elaps(t));
}

#endif
```

3.3 Animazioni

3.3.1 Avviare animazione su un evento

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_SWITCH

static void anim_x_cb(void * var, int32_t v)
{
    lv_obj_set_x((lv_obj_t *) var, v);
}

static void sw_event_cb(lv_event_t * e)
{
    lv_obj_t * sw = lv_event_get_target_obj(e);
    lv_obj_t * label = (lv_obj_t *) lv_event_get_user_data(e);

    if(lv_obj_has_state(sw, LV_STATE_CHECKED)) {
        lv_anim_t a;
        lv_anim_init(&a);
        lv_anim_set_var(&a, label);
        lv_anim_set_values(&a, lv_obj_get_x(label), 100);
        lv_anim_set_duration(&a, 500);
        lv_anim_set_exec_cb(&a, anim_x_cb);
        lv_anim_set_path_cb(&a, lv_anim_path_overshoot);
        lv_anim_start(&a);
    }
    else {
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_anim_t a;
        lv_anim_init(&a);
        lv_anim_set_var(&a, label);
        lv_anim_set_values(&a, lv_obj_get_x(label), -lv_obj_get_
↪width(label));
        lv_anim_set_duration(&a, 500);
        lv_anim_set_exec_cb(&a, anim_x_cb);
        lv_anim_set_path_cb(&a, lv_anim_path_ease_in);
        lv_anim_start(&a);
    }

}

/**
 * Start animation on an event
 */
void lv_example_anim_1(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Hello animations!");
    lv_obj_set_pos(label, 100, 10);

    lv_obj_t * sw = lv_switch_create(lv_screen_active());
    lv_obj_center(sw);
    lv_obj_add_state(sw, LV_STATE_CHECKED);
    lv_obj_add_event_cb(sw, sw_event_cb, LV_EVENT_VALUE_CHANGED, label);
}

#endif
```

3.3.2 Riproduzione di animazione

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_SWITCH

static void anim_x_cb(void * var, int32_t v)
{
    lv_obj_set_x((lv_obj_t *) var, v);
}

static void anim_size_cb(void * var, int32_t v)
{
    lv_obj_set_size((lv_obj_t *) var, v, v);
}

/**
 * Create a playback animation
 */
void lv_example_anim_2(void)
{

    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_remove_flag(obj, LV_OBJ_FLAG_SCROLLABLE);
    lv_obj_set_style_bg_color(obj, lv_palette_main(LV_PALETTE_RED), 0);
    lv_obj_set_style_radius(obj, LV_RADIUS_CIRCLE, 0);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_align(obj, LV_ALIGN_LEFT_MID, 10, 0);

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, obj);
    lv_anim_set_values(&a, 10, 50);
    lv_anim_set_duration(&a, 1000);
    lv_anim_set_reverse_delay(&a, 100);
    lv_anim_set_reverse_duration(&a, 300);
    lv_anim_set_repeat_delay(&a, 500);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);
    lv_anim_set_path_cb(&a, lv_anim_path_ease_in_out);

    lv_anim_set_exec_cb(&a, anim_size_cb);
    lv_anim_start(&a);
    lv_anim_set_exec_cb(&a, anim_x_cb);
    lv_anim_set_values(&a, 10, 240);
    lv_anim_start(&a);
}

#endif
```

3.3.3 Animazione di cubica di Bézier

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_SLIDER && LV_USE_CHART && LV_USE_BUTTON &&␣
↪LV_USE_GRID

/**
 * the example show the use of cubic-bezier3 in animation.
 * the control point P1,P2 of cubic-bezier3 can be adjusted by slider.
 * and the chart shows the cubic-bezier3 in real time. you can click
 * run button see animation in current point of cubic-bezier3.
 */

#define CHART_POINTS_NUM 256

struct {
    lv_obj_t * anim_obj;
    lv_obj_t * chart;
    lv_chart_series_t * ser1;
    lv_obj_t * p1_slider;
    lv_obj_t * p1_label;
    lv_obj_t * p2_slider;
    lv_obj_t * p2_label;
    lv_obj_t * run_btn;
    int32_t p1;
    int32_t p2;
    lv_anim_t a;
} ginfo;

static int32_t anim_path_bezier3_cb(const lv_anim_t * a);
static void refer_chart_cubic_bezier(void);
static void run_button_event_handler(lv_event_t * e);
```

(continues on next page)

(continua dalla pagina precedente)

```
static void slider_event_cb(lv_event_t * e);
static void page_obj_init(lv_obj_t * par);
static void anim_x_cb(void * var, int32_t v);

/**
 * create an animation
 */
void lv_example_anim_3(void)
{
    static int32_t col_dsc[] = {LV_GRID_FR(1), 200, LV_GRID_FR(1), LV_GRID_
↪TEMPLATE_LAST};
    static int32_t row_dsc[] = {30, 10, 10, LV_GRID_FR(1), LV_GRID_
↪TEMPLATE_LAST};

    /*Create a container with grid*/
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_style_pad_all(cont, 2, LV_PART_MAIN);
    lv_obj_set_style_pad_column(cont, 10, LV_PART_MAIN);
    lv_obj_set_style_pad_row(cont, 10, LV_PART_MAIN);
    lv_obj_set_grid_dsc_array(cont, col_dsc, row_dsc);
    lv_obj_set_size(cont, 320, 240);
    lv_obj_center(cont);

    page_obj_init(cont);

    lv_anim_init(&ginfo.a);
    lv_anim_set_var(&ginfo.a, ginfo.anim_obj);
    int32_t end = lv_obj_get_style_width(cont, LV_PART_MAIN) -
                  lv_obj_get_style_width(ginfo.anim_obj, LV_PART_MAIN) -␣
↪10;
    lv_anim_set_values(&ginfo.a, 5, end);
    lv_anim_set_duration(&ginfo.a, 2000);
    lv_anim_set_exec_cb(&ginfo.a, anim_x_cb);
    lv_anim_set_path_cb(&ginfo.a, anim_path_bezier3_cb);

    refer_chart_cubic_bezier();
}

static int32_t anim_path_bezier3_cb(const lv_anim_t * a)
{
    int32_t t = lv_map(a->act_time, 0, a->duration, 0, 1024);
    int32_t step = lv_bezier3(t, 0, ginfo.p1, ginfo.p2, 1024);
    int32_t new_value;
    new_value = step * (a->end_value - a->start_value);
    new_value = new_value >> 10;
    new_value += a->start_value;
    return new_value;
}

static void refer_chart_cubic_bezier(void)
{
    for(uint16_t i = 0; i <= CHART_POINTS_NUM; i ++) {
        int32_t t = i * (1024 / CHART_POINTS_NUM);
        int32_t step = lv_bezier3(t, 0, ginfo.p1, ginfo.p2, 1024);
        lv_chart_set_series_value_by_id2(ginfo.chart, ginfo.ser1, i, t,␣
↪step);
```

(continues on next page)

(continua dalla pagina precedente)

```
    }
    lv_chart_refresh(ginfo.chart);
}

static void anim_x_cb(void * var, int32_t v)
{
    lv_obj_set_style_translate_x((lv_obj_t *)var, v, LV_PART_MAIN);
}

static void run_button_event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    if(code == LV_EVENT_CLICKED) {
        lv_anim_start(&ginfo.a);
    }
}

static void slider_event_cb(lv_event_t * e)
{
    char buf[16];
    lv_obj_t * label;
    lv_obj_t * slider = lv_event_get_target_obj(e);

    if(slider == ginfo.p1_slider) {
        label = ginfo.p1_label;
        ginfo.p1 = lv_slider_get_value(slider);
        lv_snprintf(buf, sizeof(buf), "p1:%d", ginfo.p1);
    }
    else {
        label = ginfo.p2_label;
        ginfo.p2 = lv_slider_get_value(slider);
        lv_snprintf(buf, sizeof(buf), "p2:%d", ginfo.p2);
    }

    lv_label_set_text(label, buf);
    refer_chart_cubic_bezier();
}

static void page_obj_init(lv_obj_t * par)
{
    ginfo.anim_obj = lv_obj_create(par);
    lv_obj_set_size(ginfo.anim_obj, 30, 30);
    lv_obj_set_align(ginfo.anim_obj, LV_ALIGN_TOP_LEFT);
    lv_obj_remove_flag(ginfo.anim_obj, LV_OBJ_FLAG_SCROLLABLE);
    lv_obj_set_style_bg_color(ginfo.anim_obj, lv_palette_main(LV_PALETTE_
↪RED), LV_PART_MAIN);
    lv_obj_set_grid_cell(ginfo.anim_obj, LV_GRID_ALIGN_START, 0, 1, LV_
↪GRID_ALIGN_START, 0, 1);

    ginfo.p1_label = lv_label_create(par);
    ginfo.p2_label = lv_label_create(par);
    lv_label_set_text(ginfo.p1_label, "p1:0");
    lv_label_set_text(ginfo.p2_label, "p2:0");
    lv_obj_set_grid_cell(ginfo.p1_label, LV_GRID_ALIGN_START, 0, 1, LV_
↪GRID_ALIGN_START, 1, 1);
    lv_obj_set_grid_cell(ginfo.p2_label, LV_GRID_ALIGN_START, 0, 1, LV_
```

(continues on next page)

(continua dalla pagina precedente)

```
↪GRID_ALIGN_START, 2, 1);

    ginfo.p1_slider = lv_slider_create(par);
    ginfo.p2_slider = lv_slider_create(par);
    lv_slider_set_range(ginfo.p1_slider, 0, 1024);
    lv_slider_set_range(ginfo.p2_slider, 0, 1024);
    lv_obj_set_style_pad_all(ginfo.p1_slider, 2, LV_PART_KNOB);
    lv_obj_set_style_pad_all(ginfo.p2_slider, 2, LV_PART_KNOB);
    lv_obj_add_event_cb(ginfo.p1_slider, slider_event_cb, LV_EVENT_VALUE_
↪CHANGED, NULL);
    lv_obj_add_event_cb(ginfo.p2_slider, slider_event_cb, LV_EVENT_VALUE_
↪CHANGED, NULL);
    lv_obj_set_grid_cell(ginfo.p1_slider, LV_GRID_ALIGN_STRETCH, 1, 1, LV_
↪GRID_ALIGN_START, 1, 1);
    lv_obj_set_grid_cell(ginfo.p2_slider, LV_GRID_ALIGN_STRETCH, 1, 1, LV_
↪GRID_ALIGN_START, 2, 1);

    ginfo.run_btn = lv_button_create(par);
    lv_obj_add_event_cb(ginfo.run_btn, run_button_event_handler, LV_EVENT_
↪CLICKED, NULL);
    lv_obj_t * btn_label = lv_label_create(ginfo.run_btn);
    lv_label_set_text(btn_label, LV_SYMBOL_PLAY);
    lv_obj_center(btn_label);
    lv_obj_set_grid_cell(ginfo.run_btn, LV_GRID_ALIGN_STRETCH, 2, 1, LV_
↪GRID_ALIGN_STRETCH, 1, 2);

    ginfo.chart = lv_chart_create(par);
    lv_obj_set_style_pad_all(ginfo.chart, 0, LV_PART_MAIN);
    lv_obj_set_style_size(ginfo.chart, 0, 0, LV_PART_INDICATOR);
    lv_chart_set_type(ginfo.chart, LV_CHART_TYPE_SCATTER);
    ginfo.ser1 = lv_chart_add_series(ginfo.chart, lv_palette_main(LV_
↪PALETTE_RED), LV_CHART_AXIS_PRIMARY_Y);
    lv_chart_set_axis_range(ginfo.chart, LV_CHART_AXIS_PRIMARY_Y, 0, 1024);
    lv_chart_set_axis_range(ginfo.chart, LV_CHART_AXIS_PRIMARY_X, 0, 1024);
    lv_chart_set_point_count(ginfo.chart, CHART_POINTS_NUM);
    lv_obj_set_grid_cell(ginfo.chart, LV_GRID_ALIGN_STRETCH, 0, 3, LV_GRID_
↪ALIGN_STRETCH, 3, 1);
}

#endif
```

3.3.4 Pausa animazione

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_SWITCH

static void anim_x_cb(void * var, int32_t v)
{
    lv_obj_set_x((lv_obj_t *) var, v);
}

static void timer_cb(lv_timer_t * timer)
{

    lv_anim_t * anim = (lv_anim_t *) lv_timer_get_user_data(timer);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_anim_pause_for(anim, 1000);
    lv_timer_delete(timer);
}
static void sw_event_cb(lv_event_t * e)
{
    lv_obj_t * sw = lv_event_get_target_obj(e);
    lv_obj_t * label = (lv_obj_t *) lv_event_get_user_data(e);

    if(lv_obj_has_state(sw, LV_STATE_CHECKED)) {
        lv_anim_t a;
        lv_anim_init(&a);
        lv_anim_set_var(&a, label);
        lv_anim_set_values(&a, lv_obj_get_x(label), 100);
        lv_anim_set_duration(&a, 500);
        lv_anim_set_exec_cb(&a, anim_x_cb);
        lv_anim_set_path_cb(&a, lv_anim_path_overshoot);
        lv_timer_create(timer_cb, 200, lv_anim_start(&a));
    }
    else {
        lv_anim_t a;
        lv_anim_init(&a);
        lv_anim_set_var(&a, label);
        lv_anim_set_values(&a, lv_obj_get_x(label), -lv_obj_get_
↪width(label));
        lv_anim_set_duration(&a, 500);
        lv_anim_set_exec_cb(&a, anim_x_cb);
        lv_anim_set_path_cb(&a, lv_anim_path_ease_in);
        lv_timer_create(timer_cb, 200, lv_anim_start(&a));

    }

}

/**
 * Start animation on an event
 */
void lv_example_anim_4(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Hello animations!");
    lv_obj_set_pos(label, 100, 10);

    lv_obj_t * sw = lv_switch_create(lv_screen_active());
    lv_obj_center(sw);
    lv_obj_add_state(sw, LV_STATE_CHECKED);
    lv_obj_add_event_cb(sw, sw_event_cb, LV_EVENT_VALUE_CHANGED, label);
}

#endif
```

3.3.5 Timeline animazione

```
#include "../lv_examples.h"
#if LV_USE_FLEX && LV_BUILD_EXAMPLES
```

(continues on next page)

(continua dalla pagina precedente)

```
static const int32_t obj_width = 90;
static const int32_t obj_height = 70;

static void set_width(lv_anim_t * var, int32_t v)
{
    lv_obj_set_width((lv_obj_t *) var->var, v);
}

static void set_height(lv_anim_t * var, int32_t v)
{
    lv_obj_set_height((lv_obj_t *) var->var, v);
}

static void set_slider_value(lv_anim_t * var, int32_t v)
{
    lv_slider_set_value((lv_obj_t *) var->var, v, LV_ANIM_OFF);
}

static void btn_start_event_handler(lv_event_t * e)
{
    lv_obj_t * btn = lv_event_get_current_target_obj(e);
    lv_anim_timeline_t * anim_timeline = (lv_anim_timeline_t *) lv_event_
↪get_user_data(e);

    bool reverse = lv_obj_has_state(btn, LV_STATE_CHECKED);
    lv_anim_timeline_set_reverse(anim_timeline, reverse);
    lv_anim_timeline_start(anim_timeline);
}

static void btn_pause_event_handler(lv_event_t * e)
{
    lv_anim_timeline_t * anim_timeline = (lv_anim_timeline_t *) lv_event_
↪get_user_data(e);
    lv_anim_timeline_pause(anim_timeline);
}

static void slider_prg_event_handler(lv_event_t * e)
{
    lv_obj_t * slider = lv_event_get_current_target_obj(e);
    lv_anim_timeline_t * anim_timeline = (lv_anim_timeline_t *) lv_event_
↪get_user_data(e);
    int32_t progress = lv_slider_get_value(slider);
    lv_anim_timeline_set_progress(anim_timeline, (uint16_t) progress);
}

/**
 * Create an animation timeline
 */
void lv_example_anim_timeline_1(void)
{
    /* Create anim timeline */
    lv_anim_timeline_t * anim_timeline = lv_anim_timeline_create();

    lv_obj_t * par = lv_screen_active();
    lv_obj_set_flex_flow(par, LV_FLEX_FLOW_ROW);
    lv_obj_set_flex_align(par, LV_FLEX_ALIGN_SPACE_AROUND, LV_FLEX_ALIGN_
```

(continues on next page)

(continua dalla pagina precedente)

```
↪CENTER, LV_FLEX_ALIGN_CENTER);

    /* create btn_start */
    lv_obj_t * btn_start = lv_button_create(par);
    lv_obj_add_event_cb(btn_start, btn_start_event_handler, LV_EVENT_VALUE_
↪CHANGED, anim_timeline);
    lv_obj_add_flag(btn_start, LV_OBJ_FLAG_IGNORE_LAYOUT);
    lv_obj_add_flag(btn_start, LV_OBJ_FLAG_CHECKABLE);
    lv_obj_align(btn_start, LV_ALIGN_TOP_MID, -100, 20);

    lv_obj_t * label_start = lv_label_create(btn_start);
    lv_label_set_text(label_start, "Start");
    lv_obj_center(label_start);

    /* create btn_pause */
    lv_obj_t * btn_pause = lv_button_create(par);
    lv_obj_add_event_cb(btn_pause, btn_pause_event_handler, LV_EVENT_
↪CLICKED, anim_timeline);
    lv_obj_add_flag(btn_pause, LV_OBJ_FLAG_IGNORE_LAYOUT);
    lv_obj_align(btn_pause, LV_ALIGN_TOP_MID, 100, 20);

    lv_obj_t * label_pause = lv_label_create(btn_pause);
    lv_label_set_text(label_pause, "Pause");
    lv_obj_center(label_pause);

    /* create slider_prg */
    lv_obj_t * slider_prg = lv_slider_create(par);
    lv_obj_add_event_cb(slider_prg, slider_prg_event_handler, LV_EVENT_
↪VALUE_CHANGED, anim_timeline);
    lv_obj_add_flag(slider_prg, LV_OBJ_FLAG_IGNORE_LAYOUT);
    lv_obj_align(slider_prg, LV_ALIGN_BOTTOM_MID, 0, -20);
    lv_slider_set_range(slider_prg, 0, LV_ANIM_TIMELINE_PROGRESS_MAX);

    /* create 3 objects */
    lv_obj_t * obj1 = lv_obj_create(par);
    lv_obj_set_size(obj1, obj_width, obj_height);
    lv_obj_set_scrollbar_mode(obj1, LV_SCROLLBAR_MODE_OFF);

    lv_obj_t * obj2 = lv_obj_create(par);
    lv_obj_set_size(obj2, obj_width, obj_height);
    lv_obj_set_scrollbar_mode(obj2, LV_SCROLLBAR_MODE_OFF);

    lv_obj_t * obj3 = lv_obj_create(par);
    lv_obj_set_size(obj3, obj_width, obj_height);
    lv_obj_set_scrollbar_mode(obj3, LV_SCROLLBAR_MODE_OFF);

    /* anim-slider */
    lv_anim_t a_slider;
    lv_anim_init(&a_slider);
    lv_anim_set_var(&a_slider, slider_prg);
    lv_anim_set_values(&a_slider, 0, LV_ANIM_TIMELINE_PROGRESS_MAX);
    lv_anim_set_custom_exec_cb(&a_slider, set_slider_value);
    lv_anim_set_path_cb(&a_slider, lv_anim_path_linear);
    lv_anim_set_duration(&a_slider, 700);

    /* anim-obj1 */
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_anim_t a1;
    lv_anim_init(&a1);
    lv_anim_set_var(&a1, obj1);
    lv_anim_set_values(&a1, 0, obj_width);
    lv_anim_set_custom_exec_cb(&a1, set_width);
    lv_anim_set_path_cb(&a1, lv_anim_path_overshoot);
    lv_anim_set_duration(&a1, 300);

    lv_anim_t a2;
    lv_anim_init(&a2);
    lv_anim_set_var(&a2, obj1);
    lv_anim_set_values(&a2, 0, obj_height);
    lv_anim_set_custom_exec_cb(&a2, set_height);
    lv_anim_set_path_cb(&a2, lv_anim_path_ease_out);
    lv_anim_set_duration(&a2, 300);

    /* anim-obj2 */
    lv_anim_t a3;
    lv_anim_init(&a3);
    lv_anim_set_var(&a3, obj2);
    lv_anim_set_values(&a3, 0, obj_width);
    lv_anim_set_custom_exec_cb(&a3, set_width);
    lv_anim_set_path_cb(&a3, lv_anim_path_overshoot);
    lv_anim_set_duration(&a3, 300);

    lv_anim_t a4;
    lv_anim_init(&a4);
    lv_anim_set_var(&a4, obj2);
    lv_anim_set_values(&a4, 0, obj_height);
    lv_anim_set_custom_exec_cb(&a4, set_height);
    lv_anim_set_path_cb(&a4, lv_anim_path_ease_out);
    lv_anim_set_duration(&a4, 300);

    /* anim-obj3 */
    lv_anim_t a5;
    lv_anim_init(&a5);
    lv_anim_set_var(&a5, obj3);
    lv_anim_set_values(&a5, 0, obj_width);
    lv_anim_set_custom_exec_cb(&a5, set_width);
    lv_anim_set_path_cb(&a5, lv_anim_path_overshoot);
    lv_anim_set_duration(&a5, 300);

    lv_anim_t a6;
    lv_anim_init(&a6);
    lv_anim_set_var(&a6, obj3);
    lv_anim_set_values(&a6, 0, obj_height);
    lv_anim_set_custom_exec_cb(&a6, set_height);
    lv_anim_set_path_cb(&a6, lv_anim_path_ease_out);
    lv_anim_set_duration(&a6, 300);

    /* add animations to timeline */
    lv_anim_timeline_add(anim_timeline, 0, &a_slider);
    lv_anim_timeline_add(anim_timeline, 0, &a1);
    lv_anim_timeline_add(anim_timeline, 0, &a2);
    lv_anim_timeline_add(anim_timeline, 200, &a3);
    lv_anim_timeline_add(anim_timeline, 200, &a4);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_anim_timeline_add(anim_timeline, 400, &a5);
    lv_anim_timeline_add(anim_timeline, 400, &a6);

    lv_anim_timeline_set_progress(anim_timeline, LV_ANIM_TIMELINE_PROGRESS_
↪MAX);
}

#endif
```

3.4 Eventi

3.4.1 Evento clic pulsante

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_SWITCH

static void event_cb(lv_event_t * e)
{
    LV_LOG_USER("Clicked");

    static uint32_t cnt = 1;
    lv_obj_t * btn = lv_event_get_target_obj(e);
    lv_obj_t * label = lv_obj_get_child(btn, 0);
    lv_label_set_text_fmt(label, "%" LV_PRIu32, cnt);
    cnt++;
}

/**
 * Add click event to a button
 */
void lv_example_event_click(void)
{
    lv_obj_t * btn = lv_button_create(lv_screen_active());
    lv_obj_set_size(btn, 100, 50);
    lv_obj_center(btn);
    lv_obj_add_event_cb(btn, event_cb, LV_EVENT_CLICKED, NULL);

    lv_obj_t * label = lv_label_create(btn);
    lv_label_set_text(label, "Click me!");
    lv_obj_center(label);
}

#endif
```

3.4.2 Sequenze di clic

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_LABEL

static void short_click_event_cb(lv_event_t * e)
{
    LV_LOG_USER("Short clicked");

    lv_obj_t * info_label = (lv_obj_t *) lv_event_get_user_data(e);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_indev_t * indev = (lv_indev_t *) lv_event_get_param(e);
    uint8_t cnt = lv_indev_get_short_click_streak(indev);
    lv_label_set_text_fmt(info_label, "Short click streak: %u", cnt);
}

static void streak_event_cb(lv_event_t * e)
{
    lv_obj_t * btn = lv_event_get_target_obj(e);
    lv_obj_t * label = lv_obj_get_child(btn, 0);
    const char * text = (const char *) lv_event_get_user_data(e);
    lv_label_set_text(label, text);
}

void lv_example_event_streak(void)
{
    lv_obj_t * info_label = lv_label_create(lv_screen_active());
    lv_label_set_text(info_label, "No events yet");

    lv_obj_t * btn = lv_button_create(lv_screen_active());
    lv_obj_set_size(btn, 100, 50);
    lv_obj_center(btn);
    lv_obj_add_event_cb(btn, short_click_event_cb, LV_EVENT_SHORT_CLICKED,␣
↪info_label);
    lv_obj_add_event_cb(btn, streak_event_cb, LV_EVENT_SINGLE_CLICKED,␣
↪(void *) "Single clicked");
    lv_obj_add_event_cb(btn, streak_event_cb, LV_EVENT_DOUBLE_CLICKED,␣
↪(void *) "Double clicked");
    lv_obj_add_event_cb(btn, streak_event_cb, LV_EVENT_TRIPLE_CLICKED,␣
↪(void *) "Triple clicked");

    lv_obj_t * label = lv_label_create(btn);
    lv_label_set_text(label, "Click me!");
    lv_obj_center(label);
}

#endif
```

3.4.3 Gestione di eventi multipli

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_SWITCH

static void event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * label = (lv_obj_t *) lv_event_get_user_data(e);

    switch(code) {
        case LV_EVENT_PRESSED:
            lv_label_set_text(label, "The last button event:\nLV_EVENT_
↪PRESSED");
            break;
        case LV_EVENT_CLICKED:
            lv_label_set_text(label, "The last button event:\nLV_EVENT_
↪CLICKED");
```

(continues on next page)

(continua dalla pagina precedente)

```
            break;
        case LV_EVENT_LONG_PRESSED:
            lv_label_set_text(label, "The last button event:\nLV_EVENT_
↪LONG_PRESSED");
            break;
        case LV_EVENT_LONG_PRESSED_REPEAT:
            lv_label_set_text(label, "The last button event:\nLV_EVENT_
↪LONG_PRESSED_REPEAT");
            break;
        default:
            break;
    }
}

/**
 * Handle multiple events
 */
void lv_example_event_button(void)
{
    lv_obj_t * btn = lv_button_create(lv_screen_active());
    lv_obj_set_size(btn, 100, 50);
    lv_obj_center(btn);

    lv_obj_t * btn_label = lv_label_create(btn);
    lv_label_set_text(btn_label, "Click me!");
    lv_obj_center(btn_label);

    lv_obj_t * info_label = lv_label_create(lv_screen_active());
    lv_label_set_text(info_label, "The last button event:\nNone");

    lv_obj_add_event_cb(btn, event_cb, LV_EVENT_ALL, info_label);
}

#endif
```

3.4.4 Evento bubbling

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_FLEX

static void event_cb(lv_event_t * e)
{
    /*The original target of the event. Can be the buttons or the␣
↪container*/
    lv_obj_t * target = lv_event_get_target_obj(e);

    /*The current target is always the container as the event is added to␣
↪it*/
    lv_obj_t * cont = lv_event_get_current_target_obj(e);

    /*If container was clicked do nothing*/
    if(target == cont) return;

    /*Make the clicked buttons red*/
    lv_obj_set_style_bg_color(target, lv_palette_main(LV_PALETTE_RED), 0);
```

(continues on next page)

(continua dalla pagina precedente)

```
}

/**
 * Demonstrate event bubbling
 */
void lv_example_event_bubble(void)
{

    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 290, 200);
    lv_obj_center(cont);
    lv_obj_set_flex_flow(cont, LV_FLEX_FLOW_ROW_WRAP);

    uint32_t i;
    for(i = 0; i < 30; i++) {
        lv_obj_t * btn = lv_button_create(cont);
        lv_obj_set_size(btn, 70, 50);
        lv_obj_add_flag(btn, LV_OBJ_FLAG_EVENT_BUBBLE);

        lv_obj_t * label = lv_label_create(btn);
        lv_label_set_text_fmt(label, "%" LV_PRIu32, i);
        lv_obj_center(label);
    }

    lv_obj_add_event_cb(cont, event_cb, LV_EVENT_CLICKED, NULL);
}

#endif
```

3.4.5 Event trickle-down

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_FLEX

/**
 * Demonstrate event trickle
 */
void lv_example_event_trickle(void)
{
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 290, 200);
    lv_obj_center(cont);
    lv_obj_set_flex_flow(cont, LV_FLEX_FLOW_ROW_WRAP);

    static lv_style_t style_black;
    lv_style_init(&style_black);
    lv_style_set_text_color(&style_black, lv_color_white());
    lv_style_set_bg_color(&style_black, lv_color_black());

    /*Enable event trickle-down on the container*/
    lv_obj_add_flag(cont, LV_OBJ_FLAG_EVENT_TRICKLE);

    lv_obj_add_style(cont, &style_black, LV_STATE_PRESSED);

    uint32_t i;
```

(continues on next page)

(continua dalla pagina precedente)

```
    for(i = 0; i < 9; i++) {
        lv_obj_t * subcont = lv_obj_create(cont);
        lv_obj_set_size(subcont, 70, 50);
        lv_obj_t * label = lv_label_create(subcont);
        lv_label_set_text_fmt(label, "%" LV_PRIu32, i);

        /*Add style to the label when clicked*/
        lv_obj_add_style(subcont, &style_black, LV_STATE_FOCUSED);
    }
}

#endif
```

3.4.6 Evento Draw

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

#if LV_BUILD_EXAMPLES

static int32_t size = 0;
static bool size_dec = false;

static void timer_cb(lv_timer_t * timer)
{
    lv_obj_invalidate((lv_obj_t *) lv_timer_get_user_data(timer));
    if(size_dec) size--;
    else size++;

    if(size == 50) size_dec = true;
    else if(size == 0) size_dec = false;
}

static void event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *) lv_draw_task_
↪get_draw_dsc(draw_task);
    if(base_dsc->part == LV_PART_MAIN) {
        lv_draw_rect_dsc_t draw_dsc;
        lv_draw_rect_dsc_init(&draw_dsc);
        draw_dsc.bg_color = lv_color_hex(0xffaaaa);
        draw_dsc.radius = LV_RADIUS_CIRCLE;
        draw_dsc.border_color = lv_color_hex(0xff5555);
        draw_dsc.border_width = 2;
        draw_dsc.outline_color = lv_color_hex(0xff0000);
        draw_dsc.outline_pad = 3;
        draw_dsc.outline_width = 2;

        lv_area_t a;
        a.x1 = 0;
        a.y1 = 0;
        a.x2 = size;
        a.y2 = size;
        lv_area_t obj_coords;
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_obj_get_coords(obj, &obj_coords);
        lv_area_align(&obj_coords, &a, LV_ALIGN_CENTER, 0, 0);

        lv_draw_rect(base_dsc->layer, &draw_dsc, &a);
    }
}

/**
 * Demonstrate the usage of draw event
 */
void lv_example_event_draw(void)
{
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 200, 200);
    lv_obj_center(cont);
    lv_obj_add_event_cb(cont, event_cb, LV_EVENT_DRAW_TASK_ADDED, NULL);
    lv_obj_add_flag(cont, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);
    lv_timer_create(timer_cb, 30, cont);
}

#endif
```

3.5 Layout

3.5.1 Flex

Un semplice layout di riga e colonna con flexbox

```
#include "../../lv_examples.h"
#if LV_USE_FLEX && LV_BUILD_EXAMPLES

/**
 * A simple row and a column layout with flexbox
 */
void lv_example_flex_1(void)
{
    /*Create a container with ROW flex direction*/
    lv_obj_t * cont_row = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont_row, 300, 75);
    lv_obj_align(cont_row, LV_ALIGN_TOP_MID, 0, 5);
    lv_obj_set_flex_flow(cont_row, LV_FLEX_FLOW_ROW);

    /*Create a container with COLUMN flex direction*/
    lv_obj_t * cont_col = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont_col, 200, 150);
    lv_obj_align_to(cont_col, cont_row, LV_ALIGN_OUT_BOTTOM_MID, 0, 5);
    lv_obj_set_flex_flow(cont_col, LV_FLEX_FLOW_COLUMN);

    uint32_t i;
    for(i = 0; i < 10; i++) {
        lv_obj_t * obj;
        lv_obj_t * label;

        /*Add items to the row*/
        obj = lv_button_create(cont_row);
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_obj_set_size(obj, 100, LV_PCT(100));

        label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "Item: %" LV_PRIu32"", i);
        lv_obj_center(label);

        /*Add items to the column*/
        obj = lv_button_create(cont_col);
        lv_obj_set_size(obj, LV_PCT(100), LV_SIZE_CONTENT);

        label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "Item: %" LV_PRIu32, i);
        lv_obj_center(label);
    }
}

#endif
```

Disporre gli elementi in righe con avvolgimento e spaziatura uniforme

```
#include "../../lv_examples.h"
#if LV_USE_FLEX && LV_BUILD_EXAMPLES

/**
 * Arrange items in rows with wrap and place the items to get even space␣
 ↪around them.
 */
void lv_example_flex_2(void)
{
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_flex_flow(&style, LV_FLEX_FLOW_ROW_WRAP);
    lv_style_set_flex_main_place(&style, LV_FLEX_ALIGN_SPACE_EVENLY);
    lv_style_set_layout(&style, LV_LAYOUT_FLEX);

    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);
    lv_obj_add_style(cont, &style, 0);

    uint32_t i;
    for(i = 0; i < 8; i++) {
        lv_obj_t * obj = lv_obj_create(cont);
        lv_obj_set_size(obj, 70, LV_SIZE_CONTENT);
        lv_obj_add_flag(obj, LV_OBJ_FLAG_CHECKABLE);

        lv_obj_t * label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "%" LV_PRIu32, i);
        lv_obj_center(label);
    }
}

#endif
```

Dimostrare la flex grow.

```
#include "../../lv_examples.h"
#if LV_USE_FLEX && LV_BUILD_EXAMPLES

/**
 * Demonstrate flex grow.
 */
void lv_example_flex_3(void)
{
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);
    lv_obj_set_flex_flow(cont, LV_FLEX_FLOW_ROW);

    lv_obj_t * obj;
    obj = lv_obj_create(cont);
    lv_obj_set_size(obj, 40, 40);           /*Fix size*/

    obj = lv_obj_create(cont);
    lv_obj_set_height(obj, 40);
    lv_obj_set_flex_grow(obj, 1);           /*1 portion from the free␣
↪space*/

    obj = lv_obj_create(cont);
    lv_obj_set_height(obj, 40);
    lv_obj_set_flex_grow(obj, 2);           /*2 portion from the free␣
↪space*/

    obj = lv_obj_create(cont);
    lv_obj_set_size(obj, 40, 40);           /*Fix size. It is flushed to␣
↪the right by the "grow" items*/
}

#endif
```

Dimostrare la flex grow.

```
#include "../../lv_examples.h"
#if LV_USE_FLEX && LV_BUILD_EXAMPLES

/**
 * Reverse the order of flex items
 */
void lv_example_flex_4(void)
{

    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);
    lv_obj_set_flex_flow(cont, LV_FLEX_FLOW_COLUMN_REVERSE);

    uint32_t i;
    for(i = 0; i < 6; i++) {
        lv_obj_t * obj = lv_obj_create(cont);
        lv_obj_set_size(obj, 100, 50);
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_obj_t * label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "Item: %" LV_PRIu32, i);
        lv_obj_center(label);
    }
}

#endif
```

Dimostrare le proprietà di stile degli spazi tra colonne e righe

```
#include "../../lv_examples.h"
#if LV_USE_FLEX && LV_BUILD_EXAMPLES

static void row_gap_anim(void * obj, int32_t v)
{
    lv_obj_set_style_pad_row((lv_obj_t *) obj, v, 0);
}

static void column_gap_anim(void * obj, int32_t v)
{
    lv_obj_set_style_pad_column((lv_obj_t *) obj, v, 0);
}

/**
 * Demonstrate the effect of column and row gap style properties
 */
void lv_example_flex_5(void)
{
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);
    lv_obj_set_flex_flow(cont, LV_FLEX_FLOW_ROW_WRAP);

    uint32_t i;
    for(i = 0; i < 9; i++) {
        lv_obj_t * obj = lv_obj_create(cont);
        lv_obj_set_size(obj, 70, LV_SIZE_CONTENT);

        lv_obj_t * label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "%" LV_PRIu32, i);
        lv_obj_center(label);
    }

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, cont);
    lv_anim_set_values(&a, 0, 10);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);

    lv_anim_set_exec_cb(&a, row_gap_anim);
    lv_anim_set_duration(&a, 500);
    lv_anim_set_reverse_duration(&a, 500);
    lv_anim_start(&a);

    lv_anim_set_exec_cb(&a, column_gap_anim);
    lv_anim_set_duration(&a, 3000);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_anim_set_reverse_duration(&a, 3000);
    lv_anim_start(&a);
}

#endif
```

La direzione di base RTL modifica l'ordine degli elementi

```
#include "../../lv_examples.h"
#if LV_USE_FLEX && LV_BUILD_EXAMPLES

/**
 * RTL base direction changes order of the items.
 * Also demonstrate how horizontal scrolling works with RTL.
 */
void lv_example_flex_6(void)
{
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_style_base_dir(cont, LV_BASE_DIR_RTL, 0);
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);
    lv_obj_set_flex_flow(cont, LV_FLEX_FLOW_ROW_WRAP);

    uint32_t i;
    for(i = 0; i < 20; i++) {
        lv_obj_t * obj = lv_obj_create(cont);
        lv_obj_set_size(obj, 70, LV_SIZE_CONTENT);

        lv_obj_t * label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "%" LV_PRIu32, i);
        lv_obj_center(label);
    }
}
#endif
```

3.5.2 Grid

Una semplice grid

```
#include "../../lv_examples.h"
#if LV_USE_GRID && LV_BUILD_EXAMPLES

/**
 * A simple grid
 */
void lv_example_grid_1(void)
{
    static int32_t col_dsc[] = {70, 70, 70, LV_GRID_TEMPLATE_LAST};
    static int32_t row_dsc[] = {50, 50, 50, LV_GRID_TEMPLATE_LAST};

    /*Create a container with grid*/
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_style_grid_column_dsc_array(cont, col_dsc, 0);
    lv_obj_set_style_grid_row_dsc_array(cont, row_dsc, 0);
    lv_obj_set_size(cont, 300, 220);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_center(cont);
    lv_obj_set_layout(cont, LV_LAYOUT_GRID);

    lv_obj_t * label;
    lv_obj_t * obj;

    uint8_t i;
    for(i = 0; i < 9; i++) {
        uint8_t col = i % 3;
        uint8_t row = i / 3;

        obj = lv_button_create(cont);
        /*Stretch the cell horizontally and vertically too
         *Set span to 1 to make the cell 1 column/row sized*/
        lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_STRETCH, col, 1,
                             LV_GRID_ALIGN_STRETCH, row, 1);

        label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "c%d, r%d", col, row);
        lv_obj_center(label);
    }
}

#endif
```

Posizionamento e ampiezza delle celle

```
#include "../../lv_examples.h"
#if LV_USE_GRID && LV_BUILD_EXAMPLES

/**
 * Demonstrate cell placement and span
 */
void lv_example_grid_2(void)
{
    static int32_t col_dsc[] = {70, 70, 70, LV_GRID_TEMPLATE_LAST};
    static int32_t row_dsc[] = {50, 50, 50, LV_GRID_TEMPLATE_LAST};

    /*Create a container with grid*/
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_grid_dsc_array(cont, col_dsc, row_dsc);
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);

    lv_obj_t * label;
    lv_obj_t * obj;

    /*Cell to 0;0 and align to to the start (left/top) horizontally and␣
↪vertically too*/
    obj = lv_obj_create(cont);
    lv_obj_set_size(obj, LV_SIZE_CONTENT, LV_SIZE_CONTENT);
    lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_START, 0, 1,
                         LV_GRID_ALIGN_START, 0, 1);
    label = lv_label_create(obj);
    lv_label_set_text(label, "c0, r0");
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Cell to 1;0 and align to to the start (left) horizontally and center␣
↪vertically too*/
    obj = lv_obj_create(cont);
    lv_obj_set_size(obj, LV_SIZE_CONTENT, LV_SIZE_CONTENT);
    lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_START, 1, 1,
                         LV_GRID_ALIGN_CENTER, 0, 1);
    label = lv_label_create(obj);
    lv_label_set_text(label, "c1, r0");

    /*Cell to 2;0 and align to to the start (left) horizontally and end␣
↪(bottom) vertically too*/
    obj = lv_obj_create(cont);
    lv_obj_set_size(obj, LV_SIZE_CONTENT, LV_SIZE_CONTENT);
    lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_START, 2, 1,
                         LV_GRID_ALIGN_END, 0, 1);
    label = lv_label_create(obj);
    lv_label_set_text(label, "c2, r0");

    /*Cell to 1;1 but 2 column wide (span = 2).Set width and height to␣
↪stretched.*/
    obj = lv_obj_create(cont);
    lv_obj_set_size(obj, LV_SIZE_CONTENT, LV_SIZE_CONTENT);
    lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_STRETCH, 1, 2,
                         LV_GRID_ALIGN_STRETCH, 1, 1);
    label = lv_label_create(obj);
    lv_label_set_text(label, "c1-2, r1");

    /*Cell to 0;1 but 2 rows tall (span = 2).Set width and height to␣
↪stretched.*/
    obj = lv_obj_create(cont);
    lv_obj_set_size(obj, LV_SIZE_CONTENT, LV_SIZE_CONTENT);
    lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_STRETCH, 0, 1,
                         LV_GRID_ALIGN_STRETCH, 1, 2);
    label = lv_label_create(obj);
    lv_label_set_text(label, "c0\nr1-2");
}

#endif
```

La -free unit- della grid

```
#include "../../lv_examples.h"
#if LV_USE_GRID && LV_BUILD_EXAMPLES

/**
 * Demonstrate grid's "free unit"
 */
void lv_example_grid_3(void)
{
    /*Column 1: fix width 60 px
     *Column 2: 1 unit from the remaining free space
     *Column 3: 2 unit from the remaining free space*/
    static int32_t col_dsc[] = {60, LV_GRID_FR(1), LV_GRID_FR(2), LV_GRID_
↪TEMPLATE_LAST};

    /*Row 1: fix width 50 px
```

(continues on next page)

(continua dalla pagina precedente)

```
     *Row 2: 1 unit from the remaining free space
     *Row 3: fix width 50 px*/
    static int32_t row_dsc[] = {50, LV_GRID_FR(1), 50, LV_GRID_TEMPLATE_
↪LAST};

    /*Create a container with grid*/
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);
    lv_obj_set_grid_dsc_array(cont, col_dsc, row_dsc);

    lv_obj_t * label;
    lv_obj_t * obj;
    uint8_t i;
    for(i = 0; i < 9; i++) {
        uint8_t col = i % 3;
        uint8_t row = i / 3;

        obj = lv_obj_create(cont);
        /*Stretch the cell horizontally and vertically too
         *Set span to 1 to make the cell 1 column/row sized*/
        lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_STRETCH, col, 1,
                             LV_GRID_ALIGN_STRETCH, row, 1);

        label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "%d,%d", col, row);
        lv_obj_center(label);
    }
}

#endif
```

Posizionamento delle tracce

```
#include "../../lv_examples.h"
#if LV_USE_GRID && LV_BUILD_EXAMPLES

/**
 * Demonstrate track placement
 */
void lv_example_grid_4(void)
{
    static int32_t col_dsc[] = {60, 60, 60, LV_GRID_TEMPLATE_LAST};
    static int32_t row_dsc[] = {45, 45, 45, LV_GRID_TEMPLATE_LAST};

    /*Add space between the columns and move the rows to the bottom (end)*/

    /*Create a container with grid*/
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_grid_align(cont, LV_GRID_ALIGN_SPACE_BETWEEN, LV_GRID_ALIGN_
↪END);
    lv_obj_set_grid_dsc_array(cont, col_dsc, row_dsc);
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);

    lv_obj_t * label;
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_t * obj;
    uint8_t i;
    for(i = 0; i < 9; i++) {
        uint8_t col = i % 3;
        uint8_t row = i / 3;

        obj = lv_obj_create(cont);
        /*Stretch the cell horizontally and vertically too
         *Set span to 1 to make the cell 1 column/row sized*/
        lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_STRETCH, col, 1,
                             LV_GRID_ALIGN_STRETCH, row, 1);

        label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "%d,%d", col, row);
        lv_obj_center(label);
    }
}

#endif
```

Spazi tra colonne e righe

```
#include "../../lv_examples.h"
#if LV_USE_GRID && LV_BUILD_EXAMPLES

static void row_gap_anim(void * obj, int32_t v)
{
    lv_obj_set_style_pad_row((lv_obj_t *) obj, v, 0);
}

static void column_gap_anim(void * obj, int32_t v)
{
    lv_obj_set_style_pad_column((lv_obj_t *) obj, v, 0);
}

/**
 * Demonstrate column and row gap
 */
void lv_example_grid_5(void)
{

    /*60x60 cells*/
    static int32_t col_dsc[] = {60, 60, 60, LV_GRID_TEMPLATE_LAST};
    static int32_t row_dsc[] = {45, 45, 45, LV_GRID_TEMPLATE_LAST};

    /*Create a container with grid*/
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);
    lv_obj_set_grid_dsc_array(cont, col_dsc, row_dsc);

    lv_obj_t * label;
    lv_obj_t * obj;
    uint8_t i;
    for(i = 0; i < 9; i++) {
        uint8_t col = i % 3;
```

(continues on next page)

(continua dalla pagina precedente)

```
        uint8_t row = i / 3;

        obj = lv_obj_create(cont);
        lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_STRETCH, col, 1,
                             LV_GRID_ALIGN_STRETCH, row, 1);
        label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "%d,%d", col, row);
        lv_obj_center(label);
    }

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, cont);
    lv_anim_set_values(&a, 0, 10);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);

    lv_anim_set_exec_cb(&a, row_gap_anim);
    lv_anim_set_duration(&a, 500);
    lv_anim_set_reverse_duration(&a, 500);
    lv_anim_start(&a);

    lv_anim_set_exec_cb(&a, column_gap_anim);
    lv_anim_set_duration(&a, 3000);
    lv_anim_set_reverse_duration(&a, 3000);
    lv_anim_start(&a);
}

#endif
```

Direzione RTL sulla griglia

```
#include "../../lv_examples.h"
#if LV_USE_GRID && LV_BUILD_EXAMPLES

/**
 * Demonstrate RTL direction on grid
 */
void lv_example_grid_6(void)
{

    static int32_t col_dsc[] = {60, 60, 60, LV_GRID_TEMPLATE_LAST};
    static int32_t row_dsc[] = {45, 45, 45, LV_GRID_TEMPLATE_LAST};

    /*Create a container with grid*/
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);
    lv_obj_set_style_base_dir(cont, LV_BASE_DIR_RTL, 0);
    lv_obj_set_grid_dsc_array(cont, col_dsc, row_dsc);

    lv_obj_t * label;
    lv_obj_t * obj;
    uint8_t i;
    for(i = 0; i < 9; i++) {
        uint8_t col = i % 3;
        uint8_t row = i / 3;
```

(continues on next page)

(continua dalla pagina precedente)

```
        obj = lv_obj_create(cont);
        /*Stretch the cell horizontally and vertically too
         *Set span to 1 to make the cell 1 column/row sized*/
        lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_STRETCH, col, 1,
                             LV_GRID_ALIGN_STRETCH, row, 1);

        label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "%d,%d", col, row);
        lv_obj_center(label);
    }
}

#endif
```

3.6 Scrolling

3.6.1 Scrolling nidificato

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

static lv_obj_t  * panel;
static lv_obj_t  * save_button;
static lv_obj_t  * restore_button;
static int         saved_scroll_x;
static int         saved_scroll_y;

static void scroll_update_cb(lv_event_t * e);
static void button_event_cb(lv_event_t * e);

static void scroll_update_cb(lv_event_t * e)
{
    LV_UNUSED(e);

    LV_LOG("scroll info: x:%3" LV_PRId32", y:%3" LV_PRId32", top:%3" LV_
↪PRId32", "
           "bottom:%3" LV_PRId32", left:%3" LV_PRId32", right:%3" LV_PRId32
↪"\n",
           lv_obj_get_scroll_x(panel),
           lv_obj_get_scroll_y(panel),
           lv_obj_get_scroll_top(panel),
           lv_obj_get_scroll_bottom(panel),
           lv_obj_get_scroll_left(panel),
           lv_obj_get_scroll_right(panel)
          );
}

static void button_event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);

    if(obj == save_button) {
        saved_scroll_x = lv_obj_get_scroll_x(panel);
        saved_scroll_y = lv_obj_get_scroll_y(panel);
```

(continues on next page)

(continua dalla pagina precedente)

```
    }
    else {
        lv_obj_scroll_to(panel, saved_scroll_x, saved_scroll_y, LV_ANIM_
↪ON);
    }
}

/**
 * Demonstrate how scrolling appears automatically
 */
void lv_example_scroll_1(void)
{
    /*Create an object with the new style*/
    lv_obj_t * scr;
    scr = lv_screen_active();
    panel = lv_obj_create(scr);
    lv_obj_set_size(panel, 200, 200);
    lv_obj_align(panel, LV_ALIGN_CENTER, 44, 0);

    lv_obj_t * child;
    lv_obj_t * label;

    child = lv_obj_create(panel);
    lv_obj_set_pos(child, 0, 0);
    lv_obj_set_size(child, 70, 70);
    label = lv_label_create(child);
    lv_label_set_text(label, "Zero");
    lv_obj_center(label);

    child = lv_obj_create(panel);
    lv_obj_set_pos(child, 160, 80);
    lv_obj_set_size(child, 80, 80);

    lv_obj_t * child2 = lv_button_create(child);
    lv_obj_set_size(child2, 100, 50);

    label = lv_label_create(child2);
    lv_label_set_text(label, "Right");
    lv_obj_center(label);

    child = lv_obj_create(panel);
    lv_obj_set_pos(child, 40, 160);
    lv_obj_set_size(child, 100, 70);
    label = lv_label_create(child);
    lv_label_set_text(label, "Bottom");
    lv_obj_center(label);

    /* When LV_OBJ_FLAG_SCROLL_ELASTIC is cleared, scrolling does not go␣
↪past edge boundaries. */
    /* lv_obj_clear_flag(panel, LV_OBJ_FLAG_SCROLL_ELASTIC); */

    /* Call `scroll_update_cb` while panel is being scrolled. */
    lv_obj_add_event_cb(panel, scroll_update_cb, LV_EVENT_SCROLL, NULL);

    /* Set up buttons that save and restore scroll position. */
    save_button = lv_button_create(scr);
```

(continues on next page)

(continua dalla pagina precedente)

```
    restore_button = lv_button_create(scr);
    lv_obj_t * lbl;
    lbl = lv_label_create(save_button);
    lv_label_set_text_static(lbl, "Save");
    lbl = lv_label_create(restore_button);
    lv_label_set_text_static(lbl, "Restore");
    lv_obj_align_to(save_button, panel, LV_ALIGN_OUT_LEFT_MID, -10, -20);
    lv_obj_align_to(restore_button, panel, LV_ALIGN_OUT_LEFT_MID, -10, 20);
    lv_obj_add_event_cb(save_button, button_event_cb, LV_EVENT_CLICKED,␣
↪NULL);
    lv_obj_add_event_cb(restore_button, button_event_cb, LV_EVENT_CLICKED,␣
↪NULL);
}

#endif
```

3.6.2 Aggancio

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_FLEX

static void sw_event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * sw = lv_event_get_target_obj(e);

    if(code == LV_EVENT_VALUE_CHANGED) {
        lv_obj_t * list = (lv_obj_t *) lv_event_get_user_data(e);

        if(lv_obj_has_state(sw, LV_STATE_CHECKED)) lv_obj_add_flag(list,␣
↪LV_OBJ_FLAG_SCROLL_ONE);
        else lv_obj_remove_flag(list, LV_OBJ_FLAG_SCROLL_ONE);
    }
}

/**
 * Show an example to scroll snap
 */
void lv_example_scroll_2(void)
{
    lv_obj_t * panel = lv_obj_create(lv_screen_active());
    lv_obj_set_size(panel, 280, 120);
    lv_obj_set_scroll_snap_x(panel, LV_SCROLL_SNAP_CENTER);
    lv_obj_set_flex_flow(panel, LV_FLEX_FLOW_ROW);
    lv_obj_align(panel, LV_ALIGN_CENTER, 0, 20);

    uint32_t i;
    for(i = 0; i < 10; i++) {
        lv_obj_t * btn = lv_button_create(panel);
        lv_obj_set_size(btn, 150, lv_pct(100));

        lv_obj_t * label = lv_label_create(btn);
        if(i == 3) {
            lv_label_set_text_fmt(label, "Panel %" LV_PRIu32"\nno snap",␣
↪i);
```

(continues on next page)

(continua dalla pagina precedente)

```
            lv_obj_remove_flag(btn, LV_OBJ_FLAG_SNAPPABLE);
        }
        else {
            lv_label_set_text_fmt(label, "Panel %" LV_PRIu32, i);
        }

        lv_obj_center(label);
    }
    lv_obj_update_snap(panel, LV_ANIM_ON);

#if LV_USE_SWITCH
    /*Switch between "One scroll" and "Normal scroll" mode*/
    lv_obj_t * sw = lv_switch_create(lv_screen_active());
    lv_obj_align(sw, LV_ALIGN_TOP_RIGHT, -20, 10);
    lv_obj_add_event_cb(sw, sw_event_cb, LV_EVENT_ALL, panel);
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "One scroll");
    lv_obj_align_to(label, sw, LV_ALIGN_OUT_BOTTOM_MID, 0, 5);
#endif
}

#endif
```

3.6.3 Pulsante mobile

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_LIST

static uint32_t btn_cnt = 1;

static void float_button_event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * float_btn = lv_event_get_target_obj(e);

    if(code == LV_EVENT_CLICKED) {
        lv_obj_t * list = (lv_obj_t *) lv_event_get_user_data(e);
        char buf[32];
        lv_snprintf(buf, sizeof(buf), "Track %d", (int)btn_cnt);
        lv_obj_t * list_btn = lv_list_add_button(list, LV_SYMBOL_AUDIO,␣
↪buf);
        btn_cnt++;

        lv_obj_move_foreground(float_btn);

        lv_obj_scroll_to_view(list_btn, LV_ANIM_ON);
    }
}

/**
 * Create a list with a floating button
 */
void lv_example_scroll_3(void)
{
    lv_obj_t * list = lv_list_create(lv_screen_active());
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_size(list, 280, 220);
    lv_obj_center(list);

    for(btn_cnt = 1; btn_cnt <= 2; btn_cnt++) {
        char buf[32];
        lv_snprintf(buf, sizeof(buf), "Track %d", (int)btn_cnt);
        lv_list_add_button(list, LV_SYMBOL_AUDIO, buf);
    }

    lv_obj_t * float_btn = lv_button_create(list);
    lv_obj_set_size(float_btn, 50, 50);
    lv_obj_add_flag(float_btn, LV_OBJ_FLAG_FLOATING);
    lv_obj_align(float_btn, LV_ALIGN_BOTTOM_RIGHT, 0, -lv_obj_get_style_
↪pad_right(list, LV_PART_MAIN));
    lv_obj_add_event_cb(float_btn, float_button_event_cb, LV_EVENT_ALL,␣
↪list);
    lv_obj_set_style_radius(float_btn, LV_RADIUS_CIRCLE, 0);
    lv_obj_set_style_bg_image_src(float_btn, LV_SYMBOL_PLUS, 0);
    lv_obj_set_style_text_font(float_btn, lv_theme_get_font_large(float_
↪btn), 0);
}

#endif
```

3.6.4 Stile delle barre di scorrimento

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_LIST

/**
 * Styling the scrollbars
 */
void lv_example_scroll_4(void)
{
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_set_size(obj, 200, 100);
    lv_obj_center(obj);

    lv_obj_t * label = lv_label_create(obj);
    lv_label_set_text(label,
                      "Lorem ipsum dolor sit amet, consectetur adipiscing␣
↪elit.\n"
                      "Etiam dictum, tortor vestibulum lacinia laoreet, mi␣
↪neque consectetur neque, vel mattis odio dolor egestas ligula. \n"
                      "Sed vestibulum sapien nulla, id convallis ex␣
↪porttitor nec. \n"
                      "Duis et massa eu libero accumsan faucibus a in arcu.
↪ \n"
                      "Ut pulvinar odio lorem, vel tempus turpis␣
↪condimentum quis. Nam consectetur condimentum sem in auctor. \n"
                      "Sed nisl augue, venenatis in blandit et, gravida ac␣
↪tortor. \n"
                      "Etiam dapibus elementum suscipit. \n"
                      "Proin mollis sollicitudin convallis. \n"
                      "Integer dapibus tempus arcu nec viverra. \n"
```

(continues on next page)

(continua dalla pagina precedente)

```
                        "Donec molestie nulla enim, eu interdum velit␣
↪placerat quis. \n"
                        "Donec id efficitur risus, at molestie turpis. \n"
                        "Suspendisse vestibulum consectetur nunc ut commodo.␣
↪\n"
                        "Fusce molestie rhoncus nisi sit amet tincidunt. \n"
                        "Suspendisse a nunc ut magna ornare volutpat.");

    /*Remove the style of scrollbar to have clean start*/
    lv_obj_remove_style(obj, NULL, LV_PART_SCROLLBAR | LV_STATE_ANY);

    /*Create a transition the animate the some properties on state change*/
    static const lv_style_prop_t props[] = {LV_STYLE_BG_OPA, LV_STYLE_
↪WIDTH, 0};
    static lv_style_transition_dsc_t trans;
    lv_style_transition_dsc_init(&trans, props, lv_anim_path_linear, 200,␣
↪0, NULL);

    /*Create a style for the scrollbars*/
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_width(&style, 4);      /*Width of the scrollbar*/
    lv_style_set_length(&style, 20);    /*Length of the scrollbar*/
    lv_style_set_pad_right(&style, 5);  /*Space from the parallel side*/
    lv_style_set_pad_top(&style, 5);    /*Space from the perpendicular␣
↪side*/

    lv_style_set_radius(&style, 2);
    lv_style_set_bg_opa(&style, LV_OPA_70);
    lv_style_set_bg_color(&style, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_border_color(&style, lv_palette_darken(LV_PALETTE_BLUE,␣
↪3));
    lv_style_set_border_width(&style, 2);
    lv_style_set_shadow_width(&style, 8);
    lv_style_set_shadow_spread(&style, 2);
    lv_style_set_shadow_color(&style, lv_palette_darken(LV_PALETTE_BLUE,␣
↪1));

    lv_style_set_transition(&style, &trans);

    /*Make the scrollbars wider and use 100% opacity when scrolled*/
    static lv_style_t style_scrolled;
    lv_style_init(&style_scrolled);
    lv_style_set_width(&style_scrolled, 8);
    lv_style_set_bg_opa(&style_scrolled, LV_OPA_COVER);

    lv_obj_add_style(obj, &style, LV_PART_SCROLLBAR);
    lv_obj_add_style(obj, &style_scrolled, LV_PART_SCROLLBAR | LV_STATE_
↪SCROLLED);
}

#endif
```

3.6.5 Scorrimento da destra a sinistra

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_FONT_DEJAVU_16_PERSIAN_HEBREW

/**
 * Scrolling with Right To Left base direction
 */
void lv_example_scroll_5(void)
{
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_set_style_base_dir(obj, LV_BASE_DIR_RTL, 0);
    lv_obj_set_size(obj, 200, 100);
    lv_obj_center(obj);

    lv_obj_t * label = lv_label_create(obj);
    lv_label_set_text(label,
                      "میکروکُنترولر (به انگلیسی: Microcontroller) گونه‌ای␣
↪ریزپردازنده است که دارای حافظهٔ دسترسی تصادفی (RAM) و حافظهٔ فقط‌خواندنی␣
↪(ROM)، تایمر، پورت‌های ورودی و خروجی (I/O) و درگاه ترتیبی (Serial Port␣
↪پورت سریال)، درون خود تراشه است، و می‌تواند به تنهایی ابزارهای دیگر را␣
↪کنترل کند. به عبارت دیگر یک میکروکنترلر، مدار مجتمع کوچکی است که از یک␣
↪CPU کوچک و اجزای دیگری مانند تایمر، درگاه‌های ورودی و خروجی آنالوگ و␣
↪دیجیتال و حافظه تشکیل شده‌است.");
    lv_obj_set_width(label, 400);
    lv_obj_set_style_text_font(label, &lv_font_dejavu_16_persian_hebrew,␣
↪0);

}

#endif
```

3.6.6 Traslazione durante lo scorrimento

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_FLEX

static void scroll_event_cb(lv_event_t * e)
{
    lv_obj_t * cont = lv_event_get_target_obj(e);

    lv_area_t cont_a;
    lv_obj_get_coords(cont, &cont_a);
    int32_t cont_y_center = cont_a.y1 + lv_area_get_height(&cont_a) / 2;

    int32_t r = lv_obj_get_height(cont) * 7 / 10;
    int32_t i;
    int32_t child_cnt = (int32_t)lv_obj_get_child_count(cont);
    for(i = 0; i < child_cnt; i++) {
        lv_obj_t * child = lv_obj_get_child(cont, i);
        lv_area_t child_a;
        lv_obj_get_coords(child, &child_a);

        int32_t child_y_center = child_a.y1 + lv_area_get_height(&child_a)␣
↪/ 2;

        int32_t diff_y = child_y_center - cont_y_center;
```

(continues on next page)

(continua dalla pagina precedente)

```
        diff_y = LV_ABS(diff_y);

        /*Get the x of diff_y on a circle.*/
        int32_t x;
        /*If diff_y is out of the circle use the last point of the circle␣
↪(the radius)*/
        if(diff_y >= r) {
            x = r;
        }
        else {
            /*Use Pythagoras theorem to get x from radius and y*/
            uint32_t x_sqr = r * r - diff_y * diff_y;
            lv_sqrt_res_t res;
            lv_sqrt(x_sqr, &res, 0x8000);   /*Use lvgl's built in sqrt␣
↪root function*/
            x = r - res.i;
        }

        /*Translate the item by the calculated X coordinate*/
        lv_obj_set_style_translate_x(child, x, 0);

        /*Use some opacity with larger translations*/
        lv_opa_t opa = (lv_opa_t)lv_map(x, 0, r, LV_OPA_TRANSP, LV_OPA_
↪COVER);
        lv_obj_set_style_opa(child, LV_OPA_COVER - opa, 0);
    }
}

/**
 * Translate the object as they scroll
 */
void lv_example_scroll_6(void)
{
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 200, 200);
    lv_obj_center(cont);
    lv_obj_set_flex_flow(cont, LV_FLEX_FLOW_COLUMN);
    lv_obj_add_event_cb(cont, scroll_event_cb, LV_EVENT_SCROLL, NULL);
    lv_obj_set_style_radius(cont, LV_RADIUS_CIRCLE, 0);
    lv_obj_set_style_clip_corner(cont, true, 0);
    lv_obj_set_scroll_dir(cont, LV_DIR_VER);
    lv_obj_set_scroll_snap_y(cont, LV_SCROLL_SNAP_CENTER);
    lv_obj_set_scrollbar_mode(cont, LV_SCROLLBAR_MODE_OFF);

    uint32_t i;
    for(i = 0; i < 20; i++) {
        lv_obj_t * btn = lv_button_create(cont);
        lv_obj_set_width(btn, lv_pct(100));

        lv_obj_t * label = lv_label_create(btn);
        lv_label_set_text_fmt(label, "Button %" LV_PRIu32, i);
    }

    /*Update the buttons position manually for first*/
    lv_obj_send_event(cont, LV_EVENT_SCROLL, NULL);
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Be sure the fist button is in the middle*/
    lv_obj_scroll_to_view(lv_obj_get_child(cont, 0), LV_ANIM_OFF);
}

#endif
```

3.6.7 Scrolling infinito

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_FLEX

static lv_obj_t * high_label;
static lv_obj_t * low_label;
static int32_t top_num;
static int32_t bottom_num;
static bool update_scroll_running = false;

static lv_obj_t * load_item(lv_obj_t * parent, int32_t num)
{
    lv_obj_t * obj = (lv_obj_t *) lv_obj_create(parent);
    lv_obj_set_size(obj, LV_PCT(100), LV_SIZE_CONTENT);
    lv_obj_t * label = lv_label_create(obj);
    lv_label_set_text_fmt(label, "%" PRId32, num);
    return obj;
}

static void update_scroll(lv_obj_t * obj)
{
    /* do not re-enter this function when `lv_obj_scroll_by`
     * triggers this callback again.
     */
    if(update_scroll_running) return;
    update_scroll_running = true;

    int32_t top_num_original = top_num;
    int32_t bottom_num_original = bottom_num;

    /* load items we're getting close to */
    while(bottom_num > -30 && lv_obj_get_scroll_bottom(obj) < 200) {
        bottom_num -= 1;
        load_item(obj, bottom_num);
        lv_obj_update_layout(obj);
        LV_LOG_USER("loaded bottom num: %" PRId32, bottom_num);
    }
    while(top_num < 30 && lv_obj_get_scroll_top(obj) < 200) {
        top_num += 1;
        int32_t bottom_before = lv_obj_get_scroll_bottom(obj);
        lv_obj_t * new_item = load_item(obj, top_num);
        lv_obj_move_to_index(new_item, 0);
        lv_obj_update_layout(obj);
        int32_t bottom_after = lv_obj_get_scroll_bottom(obj);
        lv_obj_scroll_by(obj, 0, bottom_before - bottom_after, LV_ANIM_
↪OFF);
        LV_LOG_USER("loaded top num: %" PRId32, top_num);
    }
```

(continues on next page)

(continua dalla pagina precedente)

```
    /* delete far-away items */
    while(lv_obj_get_scroll_bottom(obj) > 600) {
        bottom_num += 1;
        lv_obj_t * child = lv_obj_get_child(obj, -1);
        lv_obj_delete(child);
        lv_obj_update_layout(obj);
        LV_LOG_USER("deleted bottom num: %" PRId32, bottom_num);
    }
    while(lv_obj_get_scroll_top(obj) > 600) {
        top_num -= 1;
        int32_t bottom_before = lv_obj_get_scroll_bottom(obj);
        lv_obj_t * child = lv_obj_get_child(obj, 0);
        lv_obj_delete(child);
        lv_obj_update_layout(obj);
        int32_t bottom_after = lv_obj_get_scroll_bottom(obj);
        lv_obj_scroll_by(obj, 0, bottom_before - bottom_after, LV_ANIM_
↪OFF);
        LV_LOG_USER("deleted top num: %" PRId32, top_num);
    }

    if(top_num != top_num_original) {
        lv_label_set_text_fmt(high_label, "current largest\nloaded value:\n
↪%" PRId32, top_num);
    }
    if(bottom_num != bottom_num_original) {
        lv_label_set_text_fmt(low_label, "current smallest\nloaded value:\n
↪%" PRId32, bottom_num);
    }

    update_scroll_running = false;
}

static void scroll_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    update_scroll(obj);
}

static void checkbox_cb(lv_event_t * e)
{
    lv_obj_t * checkbox = lv_event_get_target_obj(e);
    lv_obj_t * obj = (lv_obj_t *) lv_event_get_user_data(e);
    bool checked = lv_obj_has_state(checkbox, LV_STATE_CHECKED);
    lv_obj_set_style_opa(obj, checked ? LV_OPA_COVER : LV_OPA_TRANSP, LV_
↪PART_SCROLLBAR);
}

/**
 * Dynamically load widgets while scrolling
 */
void lv_example_scroll_7(void)
{
    lv_obj_t * scr = lv_screen_active();
    lv_obj_t * obj = lv_obj_create(scr);
    lv_obj_set_size(obj, 160, 220);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_align(obj, LV_ALIGN_RIGHT_MID, -10, 0);
    lv_obj_set_flex_flow(obj, LV_FLEX_FLOW_COLUMN);
    lv_obj_set_style_opa(obj, LV_OPA_TRANSP, LV_PART_SCROLLBAR);

    high_label = lv_label_create(scr);
    lv_label_set_text_static(high_label, "current largest\nloaded value:");
    lv_obj_align(high_label, LV_ALIGN_TOP_LEFT, 10, 10);

    lv_obj_t * checkbox = lv_checkbox_create(scr);
    lv_checkbox_set_text_static(checkbox, "show\nscrollbar");
    lv_obj_align(checkbox, LV_ALIGN_LEFT_MID, 10, 0);
    lv_obj_add_event_cb(checkbox, checkbox_cb, LV_EVENT_VALUE_CHANGED,␣
↪obj);

    low_label = lv_label_create(scr);
    lv_label_set_text_static(low_label, "current smallest\nloaded value:");
    lv_obj_align(low_label, LV_ALIGN_BOTTOM_LEFT, 10, -10);

    load_item(obj, 3);
    /* These counters hold the the highest/lowest number currently loaded.␣
↪*/
    top_num = 3;
    bottom_num = 3;

    lv_obj_update_layout(obj);
    update_scroll(obj);
    lv_obj_add_event_cb(obj, scroll_cb, LV_EVENT_SCROLL, NULL);
}

#endif
```

3.6.8 Scrolling circolare

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_FLEX


/*
 * Circular list implementation based on:
 * Reference: https://blog.csdn.net/TQW4321/article/details/145434819
 */

/* Applies to both width (for row items) and height (for column items) */
#define ITEM_SIZE     80

/* Function to calculate the total content width of the container */
static int32_t get_content_width(lv_obj_t * cont)
{
    int32_t w = 0;
    int32_t pad_column = lv_obj_get_style_pad_column(cont, LV_PART_MAIN); /
↪/ □□□
    int32_t total = (int32_t)lv_obj_get_child_count(cont);

    for(int32_t i = 0; i < total; i++) {
        w += lv_obj_get_width(lv_obj_get_child(cont, i));
```

(continues on next page)

(continua dalla pagina precedente)

```
        if(i < total - 1) w += pad_column;
    }
    return w + lv_obj_get_style_pad_left(cont, LV_PART_MAIN)
           + lv_obj_get_style_pad_right(cont, LV_PART_MAIN);
}

/* Scroll event callback for row layout */
static void cont_row_scroll_event_cb(lv_event_t * e)
{
    static bool is_adjusting = false;
    lv_obj_t * cont = lv_event_get_current_target_obj(e);

    if(!is_adjusting) {
        is_adjusting = true;
        int32_t scroll_x = lv_obj_get_scroll_x(cont);
        int32_t cont_w = lv_obj_get_width(cont);
        int32_t content_w = get_content_width(cont);

        /* Use ITEM_SIZE as horizontal item width */
        const int32_t item_width = ITEM_SIZE;

        if(scroll_x <= 0) {
            lv_obj_t * last_child = lv_obj_get_child(cont, (int32_t)(lv_
↪obj_get_child_count(cont) - 1));
            lv_obj_move_to_index(last_child, 0);
            lv_obj_scroll_to_x(cont, scroll_x + item_width, LV_ANIM_OFF);
        }
        else if(scroll_x > content_w - cont_w) {
            lv_obj_t * first_child = lv_obj_get_child(cont, 0);
            lv_obj_move_to_index(first_child, (int32_t)(lv_obj_get_child_
↪count(cont) - 1));
            lv_obj_scroll_to_x(cont, scroll_x - item_width, LV_ANIM_OFF);
        }
        is_adjusting = false;
    }
}

/* Function to calculate the total content height of the container */
static int32_t get_content_height(lv_obj_t * cont)
{
    int32_t h = 0;
    int32_t pad_row = lv_obj_get_style_pad_row(cont, LV_PART_MAIN);
    int32_t total = (int32_t)lv_obj_get_child_count(cont);

    for(int i = 0; i < total; i++) {
        h += lv_obj_get_height(lv_obj_get_child(cont, i));
        if(i < total - 1) h += pad_row;
    }
    return h + lv_obj_get_style_pad_top(cont, LV_PART_MAIN)
           + lv_obj_get_style_pad_bottom(cont, LV_PART_MAIN);
}

/* Scroll event callback for column layout */
static void cont_col_scroll_event_cb(lv_event_t * e)
{
    static bool is_adjusting = false;
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_t * cont = lv_event_get_current_target_obj(e);

    if(!is_adjusting) {
        is_adjusting = true;
        int32_t scroll_y = lv_obj_get_scroll_y(cont);
        int32_t cont_h = lv_obj_get_height(cont);
        int32_t content_h = get_content_height(cont);

        /* Use ITEM_SIZE as vertical item height */
        const int32_t item_height = ITEM_SIZE;

        if(scroll_y <= 0) {
            lv_obj_t * last_child = lv_obj_get_child(cont, (int32_t)(lv_
↪obj_get_child_count(cont) - 1));
            lv_obj_move_to_index(last_child, 0);
            lv_obj_scroll_to_y(cont, scroll_y + item_height, LV_ANIM_OFF);
        }
        else if(scroll_y > content_h - cont_h) {
            lv_obj_t * first_child = lv_obj_get_child(cont, 0);
            lv_obj_move_to_index(first_child, (int32_t)(lv_obj_get_child_
↪count(cont) - 1));
            lv_obj_scroll_to_y(cont, scroll_y - item_height, LV_ANIM_OFF);
        }
        is_adjusting = false;
    }
}

void lv_example_scroll_8(void)
{
    /* Create a scroll container with ROW flex direction */
    lv_obj_t * cont_row = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont_row, 300, 75);
    lv_obj_align(cont_row, LV_ALIGN_TOP_MID, 0, 5);
    lv_obj_set_flex_flow(cont_row, LV_FLEX_FLOW_ROW);
    lv_obj_add_event_cb(cont_row, cont_row_scroll_event_cb, LV_EVENT_
↪SCROLL, NULL);
    /* Hide scrollbar visuals */
    lv_obj_set_scrollbar_mode(cont_row, LV_SCROLLBAR_MODE_OFF);

    /* Create a scroll container with COLUMN flex direction */
    lv_obj_t * cont_col = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont_col, 200, 150);
    lv_obj_align_to(cont_col, cont_row, LV_ALIGN_OUT_BOTTOM_MID, 0, 5);
    lv_obj_set_flex_flow(cont_col, LV_FLEX_FLOW_COLUMN);
    lv_obj_add_event_cb(cont_col, cont_col_scroll_event_cb, LV_EVENT_
↪SCROLL, NULL);
    /* Hide scrollbar visuals */
    lv_obj_set_scrollbar_mode(cont_col, LV_SCROLLBAR_MODE_OFF);

    /* If the number of items is less than 3, scrolling may not be needed␣
↪or may cause unexpected behavior. */
    uint32_t i;
    for(i = 0; i < 10; i++) {
        lv_obj_t * obj;
        lv_obj_t * label;
```

(continues on next page)

(continua dalla pagina precedente)

```
        /*Add items to the row*/
        obj = lv_button_create(cont_row);
        lv_obj_set_size(obj, ITEM_SIZE, LV_PCT(100));

        label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "Item %" LV_PRIu32, i + 1);
        lv_obj_center(label);

        /*Add items to the column*/
        obj = lv_button_create(cont_col);
        lv_obj_set_size(obj, LV_PCT(100), ITEM_SIZE);

        label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "Item %" LV_PRIu32, i + 1);
        lv_obj_center(label);
    }
}


#endif
```

3.7 I Widget

3.7.1 Widget Base

Oggetti base con stili personalizzati

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

void lv_example_obj_1(void)
{
    lv_obj_t * obj1;
    obj1 = lv_obj_create(lv_screen_active());
    lv_obj_set_size(obj1, 100, 50);
    lv_obj_align(obj1, LV_ALIGN_CENTER, -60, -30);

    static lv_style_t style_shadow;
    lv_style_init(&style_shadow);
    lv_style_set_shadow_width(&style_shadow, 10);
    lv_style_set_shadow_spread(&style_shadow, 5);
    lv_style_set_shadow_color(&style_shadow, lv_palette_main(LV_PALETTE_
↪BLUE));

    lv_obj_t * obj2;
    obj2 = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj2, &style_shadow, 0);
    lv_obj_align(obj2, LV_ALIGN_CENTER, 60, 30);
}
#endif
```

Creare un oggetto trascinabile

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

static void drag_event_handler(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);

    lv_indev_t * indev = lv_indev_active();
    if(indev == NULL)  return;

    lv_point_t vect;
    lv_indev_get_vect(indev, &vect);

    int32_t x = lv_obj_get_x_aligned(obj) + vect.x;
    int32_t y = lv_obj_get_y_aligned(obj) + vect.y;
    lv_obj_set_pos(obj, x, y);
}

/**
 * Make an object draggable.
 */
void lv_example_obj_2(void)
{
    lv_obj_t * obj;
    obj = lv_obj_create(lv_screen_active());
    lv_obj_set_size(obj, 150, 100);
    lv_obj_add_event_cb(obj, drag_event_handler, LV_EVENT_PRESSING, NULL);

    lv_obj_t * label = lv_label_create(obj);
    lv_label_set_text(label, "Drag me");
    lv_obj_center(label);

}
#endif
```

Trasformare un oggetto usando una matrice 3x3

```
#include "../../lv_examples.h"
#if LV_BUILD_EXAMPLES
#if LV_DRAW_TRANSFORM_USE_MATRIX

static void timer_cb(lv_timer_t * timer)
{
    lv_obj_t * obj = (lv_obj_t *) lv_timer_get_user_data(timer);

    static float value = 0.1f;
    lv_matrix_t matrix;
    lv_matrix_identity(&matrix);
    lv_matrix_scale(&matrix, value, 1);
    lv_matrix_rotate(&matrix, value * 360);
    lv_obj_set_transform(obj, &matrix);

    value += 0.01f;

    if(value > 2.0f) {
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_obj_reset_transform(obj);
        value = 0.1f;
    }
}

void lv_example_obj_3(void)
{
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_center(obj);

    lv_timer_create(timer_cb, 20, obj);
}

#else

void lv_example_obj_3(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text_static(label, "LV_DRAW_TRANSFORM_USE_MATRIX is not␣
↪enabled");
    lv_obj_center(label);
}

#endif /*LV_DRAW_TRANSFORM_USE_MATRIX*/
#endif /*LV_BUILD_EXAMPLES*/
```

3.7.2 Immagine animata

Immagine animata semplice

```
#include "../../lv_examples.h"
#if LV_USE_ANIMIMG && LV_BUILD_EXAMPLES
LV_IMAGE_DECLARE(animimg001);
LV_IMAGE_DECLARE(animimg002);
LV_IMAGE_DECLARE(animimg003);

static const lv_image_dsc_t * anim_imgs[3] = {
    &animimg001,
    & animimg002,
    & animimg003,
};

void lv_example_animimg_1(void)
{
    lv_obj_t * animimg0 = lv_animimg_create(lv_screen_active());
    lv_obj_center(animimg0);
    lv_animimg_set_src(animimg0, (const void **) anim_imgs, 3);
    lv_animimg_set_duration(animimg0, 1000);
    lv_animimg_set_repeat_count(animimg0, LV_ANIM_REPEAT_INFINITE);
    lv_animimg_start(animimg0);
}

#endif
```

3.7.3 Arco

Arco semplice

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

#if LV_USE_ARC && LV_BUILD_EXAMPLES

static void value_changed_event_cb(lv_event_t * e);

void lv_example_arc_1(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());

    /*Create an Arc*/
    lv_obj_t * arc = lv_arc_create(lv_screen_active());
    lv_obj_set_size(arc, 150, 150);
    lv_arc_set_rotation(arc, 135);
    lv_arc_set_bg_angles(arc, 0, 270);
    lv_arc_set_value(arc, 10);
    lv_obj_center(arc);
    lv_obj_add_event_cb(arc, value_changed_event_cb, LV_EVENT_VALUE_
↪CHANGED, label);

    /*Manually update the label for the first time*/
    lv_obj_send_event(arc, LV_EVENT_VALUE_CHANGED, NULL);
}

static void value_changed_event_cb(lv_event_t * e)
{
    lv_obj_t * arc = lv_event_get_target_obj(e);
    lv_obj_t * label = (lv_obj_t *)lv_event_get_user_data(e);

    lv_label_set_text_fmt(label, "%" LV_PRId32 "%%", lv_arc_get_
↪value(arc));

    /*Rotate the label to the current position of the arc*/
    lv_arc_rotate_obj_to_angle(arc, label, 25);
}

#endif
```

Loader con Arc

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

#if LV_USE_ARC && LV_BUILD_EXAMPLES

static void set_angle(void * obj, int32_t v)
{
    lv_arc_set_value((lv_obj_t *)obj, v);
}

/**
 * Create an arc which acts as a loader.
 */
void lv_example_arc_2(void)
```

(continues on next page)

(continua dalla pagina precedente)

```
{
    /*Create an Arc*/
    lv_obj_t * arc = lv_arc_create(lv_screen_active());
    lv_arc_set_rotation(arc, 270);
    lv_arc_set_bg_angles(arc, 0, 360);
    lv_obj_remove_style(arc, NULL, LV_PART_KNOB);   /*Be sure the knob is␣
↪not displayed*/
    lv_obj_remove_flag(arc, LV_OBJ_FLAG_CLICKABLE);  /*To not allow␣
↪adjusting by click*/
    lv_obj_center(arc);

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, arc);
    lv_anim_set_exec_cb(&a, set_angle);
    lv_anim_set_duration(&a, 1000);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);    /*Just for␣
↪the demo*/
    lv_anim_set_repeat_delay(&a, 500);
    lv_anim_set_values(&a, 0, 100);
    lv_anim_start(&a);

}

#endif
```

Grafico a torta con sezioni cliccabili tramite Archi

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

#if LV_USE_ARC && LV_BUILD_EXAMPLES

#define CHART_SIZE 150
#define SLICE_OFFSET 20

typedef struct {
    int start_angle;
    int end_angle;
    int mid_angle;
    lv_point_t home;
    bool out;
} slice_info_t;

typedef struct {
    lv_obj_t * obj;
    int start_x;
    int start_y;
    int end_x;
    int end_y;
} slice_anim_data_t;

static float angle_accum = 0.0f;
static slice_info_t * active_info = NULL;
static lv_obj_t * active_arc = NULL;

static void anim_move_cb(void * var, int32_t v)
```

(continues on next page)

(continua dalla pagina precedente)

```
{
    slice_anim_data_t * d = (slice_anim_data_t *) var;

    int32_t x = d->start_x + ((d->end_x - d->start_x) * v) / 100;
    int32_t y = d->start_y + ((d->end_y - d->start_y) * v) / 100;
    lv_obj_set_pos(d->obj, x, y);
}

static void anim_cleanup_cb(lv_anim_t * a)
{
    lv_free(a->var);
}

static void arc_click_cb(lv_event_t * e)
{
    lv_obj_t * arc  = lv_event_get_target_obj(e);
    slice_info_t * info = (slice_info_t *)lv_event_get_user_data(e);

    int32_t x_off = (SLICE_OFFSET * lv_trigo_cos(info->mid_angle)) >> LV_
↪TRIGO_SHIFT;
    int32_t y_off = (SLICE_OFFSET * lv_trigo_sin(info->mid_angle)) >> LV_
↪TRIGO_SHIFT;

    if(active_info && active_info != info && active_info->out) {
        slice_anim_data_t * anim_back = (slice_anim_data_t *) lv_
↪malloc(sizeof(slice_anim_data_t));
        anim_back->obj = active_arc;
        anim_back->start_x = lv_obj_get_x(active_arc) - SLICE_OFFSET;
        anim_back->start_y = lv_obj_get_y(active_arc) - SLICE_OFFSET;
        anim_back->end_x = active_info->home.x;
        anim_back->end_y = active_info->home.y;

        active_info->out = false;

        lv_anim_t a;
        lv_anim_init(&a);
        lv_anim_set_var(&a, anim_back);
        lv_anim_set_exec_cb(&a, anim_move_cb);
        lv_anim_set_time(&a, 200);
        lv_anim_set_values(&a, 0, 100);
        lv_anim_set_deleted_cb(&a, anim_cleanup_cb);
        lv_anim_start(&a);
    }

    int target_x, target_y;
    if(info->out) {
        target_x  = info->home.x;
        target_y  = info->home.y;
        info->out = false;
        active_info = NULL;
        active_arc = NULL;
    }
    else {
        target_x  = info->home.x + x_off;
        target_y  = info->home.y + y_off;
        info->out = true;
```

(continues on next page)

(continua dalla pagina precedente)

```
        active_info = info;
        active_arc = arc;
    }

    slice_anim_data_t * anim_data = (slice_anim_data_t *) lv_
↪malloc(sizeof(slice_anim_data_t));
    anim_data->obj     = arc;
    anim_data->start_x = lv_obj_get_x(arc) - SLICE_OFFSET;
    anim_data->start_y = lv_obj_get_y(arc) - SLICE_OFFSET;
    anim_data->end_x   = target_x;
    anim_data->end_y   = target_y;

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, anim_data);
    lv_anim_set_exec_cb(&a, anim_move_cb);
    lv_anim_set_time(&a, 200);
    lv_anim_set_values(&a, 0, 100);
    lv_anim_set_deleted_cb(&a, anim_cleanup_cb);
    lv_anim_start(&a);
}

static void create_slice(lv_obj_t * parent, int percentage, lv_color_t␣
↪color)
{
    if(percentage <= 0) return;

    float slice_angle = (percentage * 360.0f) / 100.0f;
    int start = (int)(angle_accum + 0.5f);
    angle_accum += slice_angle;
    int end = (int)(angle_accum + 0.5f);
    if(end > 360) end = 360;

    lv_obj_t * arc = lv_arc_create(parent);
    lv_obj_set_size(arc, CHART_SIZE, CHART_SIZE);
    lv_obj_center(arc);

    lv_arc_set_mode(arc, LV_ARC_MODE_NORMAL);
    lv_arc_set_bg_start_angle(arc, start);
    lv_arc_set_bg_end_angle(arc, end);

    lv_obj_set_style_arc_width(arc, CHART_SIZE / 2, LV_PART_MAIN);
    lv_obj_set_style_arc_width(arc, 0, LV_PART_INDICATOR);

    lv_obj_set_style_arc_color(arc, color, LV_PART_MAIN);
    lv_obj_set_style_arc_rounded(arc, false, LV_PART_MAIN);
    lv_obj_remove_style(arc, NULL, LV_PART_KNOB);
    lv_obj_add_flag(arc, LV_OBJ_FLAG_ADV_HITTEST);

    lv_obj_t * label = lv_label_create(arc);
    lv_label_set_text_fmt(label, "%d%%", percentage);
    int mid_angle = start + ((end - start) / 2);
    int radius = CHART_SIZE / 4;
    int x_offset = (radius * lv_trigo_cos(mid_angle)) >> LV_TRIGO_SHIFT;
    int y_offset = (radius * lv_trigo_sin(mid_angle)) >> LV_TRIGO_SHIFT;
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_align(label, LV_ALIGN_CENTER, x_offset, y_offset);

    slice_info_t * info = (slice_info_t *) lv_malloc(sizeof(slice_info_t));
    info->start_angle = start;
    info->end_angle = end;
    info->mid_angle = mid_angle;
    info->out         = false;
    info->home.x = lv_obj_get_x(arc);
    info->home.y = lv_obj_get_y(arc);
    lv_obj_add_event_cb(arc, arc_click_cb, LV_EVENT_CLICKED, info);
}

void lv_example_arc_3(void)
{
    /* Root container: flex row */
    lv_obj_t * root = lv_obj_create(lv_screen_active());
    lv_obj_set_size(root, LV_SIZE_CONTENT, LV_SIZE_CONTENT);
    lv_obj_center(root);
    lv_obj_set_flex_flow(root, LV_FLEX_FLOW_ROW);
    lv_obj_set_flex_align(root, LV_FLEX_ALIGN_CENTER, LV_FLEX_ALIGN_CENTER,
↪ LV_FLEX_ALIGN_CENTER);

    lv_obj_set_style_pad_all(root, 0, LV_PART_MAIN);
    lv_obj_set_style_border_width(root, 0, LV_PART_MAIN);
    lv_obj_set_style_border_color(root, lv_palette_main(LV_PALETTE_RED),␣
↪LV_PART_MAIN);
    lv_obj_set_style_bg_opa(root, LV_OPA_TRANSP, LV_PART_MAIN);
    lv_obj_remove_flag(root, LV_OBJ_FLAG_SCROLLABLE);

    /* Slices container */
    lv_obj_t * slices_container = lv_obj_create(root);
    lv_obj_set_size(slices_container, CHART_SIZE + 2 * SLICE_OFFSET, CHART_
↪SIZE + 2 * SLICE_OFFSET);
    lv_obj_set_style_pad_all(slices_container, 0, LV_PART_MAIN);
    lv_obj_set_style_margin_all(slices_container, 0, LV_PART_MAIN);
    lv_obj_set_style_border_width(slices_container, 0, LV_PART_MAIN);
    lv_obj_set_style_border_color(slices_container, lv_palette_main(LV_
↪PALETTE_BLUE), LV_PART_MAIN);
    lv_obj_set_style_bg_opa(slices_container, LV_OPA_TRANSP, LV_PART_MAIN);
    lv_obj_remove_flag(slices_container, LV_OBJ_FLAG_SCROLLABLE);

    /* Create slices */
    angle_accum = 0.0f;
    create_slice(slices_container, 12, lv_palette_main(LV_PALETTE_RED));
    create_slice(slices_container, 18, lv_palette_main(LV_PALETTE_BLUE));
    create_slice(slices_container, 26, lv_palette_main(LV_PALETTE_GREEN));
    create_slice(slices_container, 24, lv_palette_main(LV_PALETTE_ORANGE));
    create_slice(slices_container, 20, lv_palette_main(LV_PALETTE_BLUE_
↪GREY));
}

#endif
```

3.7.4 Barra

Barra semplice

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES

void lv_example_bar_1(void)
{
    lv_obj_t * bar1 = lv_bar_create(lv_screen_active());
    lv_obj_set_size(bar1, 200, 20);
    lv_obj_center(bar1);
    lv_bar_set_value(bar1, 70, LV_ANIM_OFF);
}

#endif
```

Stile di una barra

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES

/**
 * Example of styling the bar
 */
void lv_example_bar_2(void)
{
    static lv_style_t style_bg;
    static lv_style_t style_indic;

    lv_style_init(&style_bg);
    lv_style_set_border_color(&style_bg, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_border_width(&style_bg, 2);
    lv_style_set_pad_all(&style_bg, 6); /*To make the indicator smaller*/
    lv_style_set_radius(&style_bg, 6);
    lv_style_set_anim_duration(&style_bg, 1000);

    lv_style_init(&style_indic);
    lv_style_set_bg_opa(&style_indic, LV_OPA_COVER);
    lv_style_set_bg_color(&style_indic, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_radius(&style_indic, 3);

    lv_obj_t * bar = lv_bar_create(lv_screen_active());
    lv_obj_remove_style_all(bar);  /*To have a clean start*/
    lv_obj_add_style(bar, &style_bg, 0);
    lv_obj_add_style(bar, &style_indic, LV_PART_INDICATOR);

    lv_obj_set_size(bar, 200, 20);
    lv_obj_center(bar);
    lv_bar_set_value(bar, 100, LV_ANIM_ON);
}

#endif
```

Indicatore di temperatura

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES

static void set_temp(void * bar, int32_t temp)
{
    lv_bar_set_value((lv_obj_t *)bar, temp, LV_ANIM_ON);
}

/**
 * A temperature meter example
 */
void lv_example_bar_3(void)
{
    static lv_style_t style_indic;

    lv_style_init(&style_indic);
    lv_style_set_bg_opa(&style_indic, LV_OPA_COVER);
    lv_style_set_bg_color(&style_indic, lv_palette_main(LV_PALETTE_RED));
    lv_style_set_bg_grad_color(&style_indic, lv_palette_main(LV_PALETTE_
↪BLUE));
    lv_style_set_bg_grad_dir(&style_indic, LV_GRAD_DIR_VER);

    lv_obj_t * bar = lv_bar_create(lv_screen_active());
    lv_obj_add_style(bar, &style_indic, LV_PART_INDICATOR);
    lv_obj_set_size(bar, 20, 200);
    lv_obj_center(bar);
    lv_bar_set_range(bar, -20, 40);

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_exec_cb(&a, set_temp);
    lv_anim_set_duration(&a, 3000);
    lv_anim_set_reverse_duration(&a, 3000);
    lv_anim_set_var(&a, bar);
    lv_anim_set_values(&a, -20, 40);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);
    lv_anim_start(&a);
}

#endif
```

Motivo a strisce e valore dell'intervallo

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES

/**
 * Bar with stripe pattern and ranged value
 */
void lv_example_bar_4(void)
{
    LV_IMAGE_DECLARE(img_skew_strip);
    static lv_style_t style_indic;

    lv_style_init(&style_indic);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_set_bg_image_src(&style_indic, &img_skew_strip);
    lv_style_set_bg_image_tiled(&style_indic, true);
    lv_style_set_bg_image_opa(&style_indic, LV_OPA_30);

    lv_obj_t * bar = lv_bar_create(lv_screen_active());
    lv_obj_add_style(bar, &style_indic, LV_PART_INDICATOR);

    lv_obj_set_size(bar, 260, 20);
    lv_obj_center(bar);
    lv_bar_set_mode(bar, LV_BAR_MODE_RANGE);
    lv_bar_set_value(bar, 90, LV_ANIM_OFF);
    lv_bar_set_start_value(bar, 20, LV_ANIM_OFF);
}

#endif
```

Barra con direzione di base LTR e RTL

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES

/**
 * Bar with LTR and RTL base direction
 */
void lv_example_bar_5(void)
{
    lv_obj_t * label;

    lv_obj_t * bar_ltr = lv_bar_create(lv_screen_active());
    lv_obj_set_size(bar_ltr, 200, 20);
    lv_bar_set_value(bar_ltr, 70, LV_ANIM_OFF);
    lv_obj_align(bar_ltr, LV_ALIGN_CENTER, 0, -30);

    label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Left to Right base direction");
    lv_obj_align_to(label, bar_ltr, LV_ALIGN_OUT_TOP_MID, 0, -5);

    lv_obj_t * bar_rtl = lv_bar_create(lv_screen_active());
    lv_obj_set_style_base_dir(bar_rtl, LV_BASE_DIR_RTL, 0);
    lv_obj_set_size(bar_rtl, 200, 20);
    lv_bar_set_value(bar_rtl, 70, LV_ANIM_OFF);
    lv_obj_align(bar_rtl, LV_ALIGN_CENTER, 0, 30);

    label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Right to Left base direction");
    lv_obj_align_to(label, bar_rtl, LV_ALIGN_OUT_TOP_MID, 0, -5);
}

#endif
```

Cassetto personalizzato per visualizzare il valore corrente

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES
```

(continues on next page)

(continua dalla pagina precedente)

```
#define MAX_VALUE 100
#define MIN_VALUE 0

static void set_value(void * bar, int32_t v)
{
    lv_bar_set_value((lv_obj_t *)bar, v, LV_ANIM_OFF);
}

static void event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);

    lv_draw_label_dsc_t label_dsc;
    lv_draw_label_dsc_init(&label_dsc);
    label_dsc.font = LV_FONT_DEFAULT;

    char buf[8];
    lv_snprintf(buf, sizeof(buf), "%d", (int)lv_bar_get_value(obj));

    lv_point_t txt_size;
    lv_text_get_size(&txt_size, buf, label_dsc.font, label_dsc.letter_
↪space, label_dsc.line_space, LV_COORD_MAX,
                     label_dsc.flag);

    lv_area_t txt_area;
    txt_area.x1 = 0;
    txt_area.x2 = txt_size.x - 1;
    txt_area.y1 = 0;
    txt_area.y2 = txt_size.y - 1;

    lv_area_t indic_area;
    lv_obj_get_coords(obj, &indic_area);
    lv_area_set_width(&indic_area, lv_area_get_width(&indic_area) * lv_bar_
↪get_value(obj) / MAX_VALUE);

    /*If the indicator is long enough put the text inside on the right*/
    if(lv_area_get_width(&indic_area) > txt_size.x + 20) {
        lv_area_align(&indic_area, &txt_area, LV_ALIGN_RIGHT_MID, -10, 0);
        label_dsc.color = lv_color_white();
    }
    /*If the indicator is still short put the text out of it on the right*/
    else {
        lv_area_align(&indic_area, &txt_area, LV_ALIGN_OUT_RIGHT_MID, 10,␣
↪0);
        label_dsc.color = lv_color_black();
    }
    label_dsc.text = buf;
    label_dsc.text_local = true;
    lv_layer_t * layer = lv_event_get_layer(e);
    lv_draw_label(layer, &label_dsc, &txt_area);
}

/**
 * Custom drawer on the bar to display the current value
 */
void lv_example_bar_6(void)
```

(continues on next page)

(continua dalla pagina precedente)

```
{
    lv_obj_t * bar = lv_bar_create(lv_screen_active());
    lv_bar_set_range(bar, MIN_VALUE, MAX_VALUE);
    lv_obj_set_size(bar, 200, 20);
    lv_obj_center(bar);
    lv_obj_add_event_cb(bar, event_cb, LV_EVENT_DRAW_MAIN_END, NULL);

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, bar);
    lv_anim_set_values(&a, 0, 100);
    lv_anim_set_exec_cb(&a, set_value);
    lv_anim_set_duration(&a, 4000);
    lv_anim_set_reverse_duration(&a, 4000);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);
    lv_anim_start(&a);

}

#endif
```

Barra con direzione opposta

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES

/**
 * Bar with opposite direction
 */
void lv_example_bar_7(void)
{
    lv_obj_t * bar_tob = lv_bar_create(lv_screen_active());
    lv_obj_set_size(bar_tob, 20, 180);
    lv_bar_set_range(bar_tob, 100, 0);
    lv_bar_set_value(bar_tob, 70, LV_ANIM_OFF);
    lv_obj_align(bar_tob, LV_ALIGN_CENTER, 0, 10);

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "From top to bottom");
    lv_obj_align_to(label, bar_tob, LV_ALIGN_OUT_TOP_MID, 0, -5);
}

#endif
```

3.7.5 Pulsante

Pulsanti semplici

```
#include "../../lv_examples.h"
#if LV_USE_BUTTON && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
```

(continues on next page)

(continua dalla pagina precedente)

```
    if(code == LV_EVENT_CLICKED) {
        LV_LOG_USER("Clicked");
    }
    else if(code == LV_EVENT_VALUE_CHANGED) {
        LV_LOG_USER("Toggled");
    }
}

void lv_example_button_1(void)
{
    lv_obj_t * label;

    lv_obj_t * btn1 = lv_button_create(lv_screen_active());
    lv_obj_add_event_cb(btn1, event_handler, LV_EVENT_ALL, NULL);
    lv_obj_align(btn1, LV_ALIGN_CENTER, 0, -40);
    lv_obj_remove_flag(btn1, LV_OBJ_FLAG_PRESS_LOCK);

    label = lv_label_create(btn1);
    lv_label_set_text(label, "Button");
    lv_obj_center(label);

    lv_obj_t * btn2 = lv_button_create(lv_screen_active());
    lv_obj_add_event_cb(btn2, event_handler, LV_EVENT_ALL, NULL);
    lv_obj_align(btn2, LV_ALIGN_CENTER, 0, 40);
    lv_obj_add_flag(btn2, LV_OBJ_FLAG_CHECKABLE);
    lv_obj_set_height(btn2, LV_SIZE_CONTENT);

    label = lv_label_create(btn2);
    lv_label_set_text(label, "Toggle");
    lv_obj_center(label);

}
#endif
```

Pulsanti con stili

```
#include "../../lv_examples.h"
#if LV_USE_BUTTON && LV_BUILD_EXAMPLES

/**
 * Style a button from scratch
 */
void lv_example_button_2(void)
{
    /*Init the style for the default state*/
    static lv_style_t style;
    lv_style_init(&style);

    lv_style_set_radius(&style, 3);

    lv_style_set_bg_opa(&style, LV_OPA_100);
    lv_style_set_bg_color(&style, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_bg_grad_color(&style, lv_palette_darken(LV_PALETTE_BLUE,␣
↪2));
    lv_style_set_bg_grad_dir(&style, LV_GRAD_DIR_VER);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_set_border_opa(&style, LV_OPA_40);
    lv_style_set_border_width(&style, 2);
    lv_style_set_border_color(&style, lv_palette_main(LV_PALETTE_GREY));

    lv_style_set_shadow_width(&style, 8);
    lv_style_set_shadow_color(&style, lv_palette_main(LV_PALETTE_GREY));
    lv_style_set_shadow_offset_y(&style, 8);

    lv_style_set_outline_opa(&style, LV_OPA_COVER);
    lv_style_set_outline_color(&style, lv_palette_main(LV_PALETTE_BLUE));

    lv_style_set_text_color(&style, lv_color_white());
    lv_style_set_pad_all(&style, 10);

    /*Init the pressed style*/
    static lv_style_t style_pr;
    lv_style_init(&style_pr);

    /*Add a large outline when pressed*/
    lv_style_set_outline_width(&style_pr, 30);
    lv_style_set_outline_opa(&style_pr, LV_OPA_TRANSP);

    lv_style_set_translate_y(&style_pr, 5);
    lv_style_set_shadow_offset_y(&style_pr, 3);
    lv_style_set_bg_color(&style_pr, lv_palette_darken(LV_PALETTE_BLUE,␣
↪2));
    lv_style_set_bg_grad_color(&style_pr, lv_palette_darken(LV_PALETTE_
↪BLUE, 4));

    /*Add a transition to the outline*/
    static lv_style_transition_dsc_t trans;
    static lv_style_prop_t props[] = {LV_STYLE_OUTLINE_WIDTH, LV_STYLE_
↪OUTLINE_OPA, 0};
    lv_style_transition_dsc_init(&trans, props, lv_anim_path_linear, 300,␣
↪0, NULL);

    lv_style_set_transition(&style_pr, &trans);

    lv_obj_t * btn1 = lv_button_create(lv_screen_active());
    lv_obj_remove_style_all(btn1);                          /*Remove the␣
↪style coming from the theme*/
    lv_obj_add_style(btn1, &style, 0);
    lv_obj_add_style(btn1, &style_pr, LV_STATE_PRESSED);
    lv_obj_set_size(btn1, LV_SIZE_CONTENT, LV_SIZE_CONTENT);
    lv_obj_center(btn1);

    lv_obj_t * label = lv_label_create(btn1);
    lv_label_set_text(label, "Button");
    lv_obj_center(label);
}
#endif
```

Pulsante gommoso

```
#include "../../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_BUTTON

/**
 * Create a style transition on a button to act like a gum when clicked
 */
void lv_example_button_3(void)
{
    /*Properties to transition*/
    static lv_style_prop_t props[] = {
        LV_STYLE_TRANSFORM_WIDTH, LV_STYLE_TRANSFORM_HEIGHT, LV_STYLE_TEXT_
↪LETTER_SPACE, 0
    };

    /*Transition descriptor when going back to the default state.
     *Add some delay to be sure the press transition is visible even if␣
↪the press was very short*/
    static lv_style_transition_dsc_t transition_dsc_def;
    lv_style_transition_dsc_init(&transition_dsc_def, props, lv_anim_path_
↪overshoot, 250, 100, NULL);

    /*Transition descriptor when going to pressed state.
     *No delay, go to presses state immediately*/
    static lv_style_transition_dsc_t transition_dsc_pr;
    lv_style_transition_dsc_init(&transition_dsc_pr, props, lv_anim_path_
↪ease_in_out, 250, 0, NULL);

    /*Add only the new transition to he default state*/
    static lv_style_t style_def;
    lv_style_init(&style_def);
    lv_style_set_transition(&style_def, &transition_dsc_def);

    /*Add the transition and some transformation to the presses state.*/
    static lv_style_t style_pr;
    lv_style_init(&style_pr);
    lv_style_set_transform_width(&style_pr, 10);
    lv_style_set_transform_height(&style_pr, -10);
    lv_style_set_text_letter_space(&style_pr, 10);
    lv_style_set_transition(&style_pr, &transition_dsc_pr);

    lv_obj_t * btn1 = lv_button_create(lv_screen_active());
    lv_obj_align(btn1, LV_ALIGN_CENTER, 0, -80);
    lv_obj_add_style(btn1, &style_pr, LV_STATE_PRESSED);
    lv_obj_add_style(btn1, &style_def, 0);

    lv_obj_t * label = lv_label_create(btn1);
    lv_label_set_text(label, "Gum");
}
#endif
```

3.7.6 Matrice di Pulsanti

Matrice di pulsanti semplice

```
#include "../../lv_examples.h"
#if LV_USE_BUTTONMATRIX && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        uint32_t id = lv_buttonmatrix_get_selected_button(obj);
        const char * txt = lv_buttonmatrix_get_button_text(obj, id);
        LV_UNUSED(txt);
        LV_LOG_USER("%s was pressed\n", txt);
    }
}

static const char * btnm_map[] = {"1", "2", "3", "4", "5", "\n",
                                  "6", "7", "8", "9", "0", "\n",
                                  "Action1", "Action2", ""
                                 };

void lv_example_buttonmatrix_1(void)
{
    lv_obj_t * btnm1 = lv_buttonmatrix_create(lv_screen_active());
    lv_buttonmatrix_set_map(btnm1, btnm_map);
    lv_buttonmatrix_set_button_width(btnm1, 10, 2);        /*Make "Action1
↪" twice as wide as "Action2"*/
    lv_buttonmatrix_set_button_ctrl(btnm1, 10, LV_BUTTONMATRIX_CTRL_
↪CHECKABLE);
    lv_buttonmatrix_set_button_ctrl(btnm1, 11, LV_BUTTONMATRIX_CTRL_
↪CHECKED);
    lv_obj_align(btnm1, LV_ALIGN_CENTER, 0, 0);
    lv_obj_add_event_cb(btnm1, event_handler, LV_EVENT_ALL, NULL);
}

#endif
```

Pulsanti personalizzati

```
#include "../../lv_examples.h"
#if LV_USE_BUTTONMATRIX && LV_BUILD_EXAMPLES

static void event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);
    /*When the button matrix draws the buttons...*/
    if(base_dsc->part == LV_PART_ITEMS) {
        bool pressed = false;
        if(lv_buttonmatrix_get_selected_button(obj) == base_dsc->id1 && lv_
↪obj_has_state(obj, LV_STATE_PRESSED)) {
            pressed = true;
        }

        /*Change the draw descriptor of the 2nd button*/
```

(continues on next page)

(continua dalla pagina precedente)

```
        if(base_dsc->id1 == 1) {
            lv_draw_fill_dsc_t * fill_draw_dsc = lv_draw_task_get_fill_
↪dsc(draw_task);
            if(fill_draw_dsc) {
                fill_draw_dsc->radius = 0;
                if(pressed) fill_draw_dsc->color = lv_palette_darken(LV_
↪PALETTE_BLUE, 3);
                else fill_draw_dsc->color = lv_palette_main(LV_PALETTE_
↪BLUE);
            }
            lv_draw_box_shadow_dsc_t * box_shadow_draw_dsc = lv_draw_task_
↪get_box_shadow_dsc(draw_task);
            if(box_shadow_draw_dsc) {
                box_shadow_draw_dsc->width = 6;
                box_shadow_draw_dsc->ofs_x = 3;
                box_shadow_draw_dsc->ofs_y = 3;
            }
            lv_draw_label_dsc_t * label_draw_dsc = lv_draw_task_get_label_
↪dsc(draw_task);
            if(label_draw_dsc) {
                label_draw_dsc->color = lv_color_white();
            }

        }
        /*Change the draw descriptor of the 3rd button*/
        else if(base_dsc->id1 == 2) {
            lv_draw_fill_dsc_t * fill_draw_dsc = lv_draw_task_get_fill_
↪dsc(draw_task);
            if(fill_draw_dsc) {
                fill_draw_dsc->radius = LV_RADIUS_CIRCLE;
                if(pressed) fill_draw_dsc->color = lv_palette_darken(LV_
↪PALETTE_RED, 3);
                else fill_draw_dsc->color = lv_palette_main(LV_PALETTE_
↪RED);
            }

            lv_draw_box_shadow_dsc_t * box_shadow_draw_dsc = lv_draw_task_
↪get_box_shadow_dsc(draw_task);
            if(box_shadow_draw_dsc) {
                box_shadow_draw_dsc->radius = LV_RADIUS_CIRCLE;
            }
        }
        else if(base_dsc->id1 == 3) {
            lv_draw_label_dsc_t * label_draw_dsc = lv_draw_task_get_label_
↪dsc(draw_task);
            if(label_draw_dsc) {
                label_draw_dsc->opa = 0;
            }
            if(lv_draw_task_get_type(draw_task) == LV_DRAW_TASK_TYPE_FILL)
↪{
                LV_IMAGE_DECLARE(img_star);
                lv_image_header_t header;
                lv_result_t res = lv_image_decoder_get_info(&img_star, &
↪header);
                if(res != LV_RESULT_OK) return;
```

(continues on next page)

(continua dalla pagina precedente)

```
                lv_area_t a;
                a.x1 = 0;
                a.x2 = header.w - 1;
                a.y1 = 0;
                a.y2 = header.h - 1;
                lv_area_t draw_task_area;
                lv_draw_task_get_area(draw_task, &draw_task_area);
                lv_area_align(&draw_task_area, &a, LV_ALIGN_CENTER, 0, 0);

                lv_draw_image_dsc_t img_draw_dsc;
                lv_draw_image_dsc_init(&img_draw_dsc);
                img_draw_dsc.src = &img_star;
                img_draw_dsc.recolor = lv_color_black();
                if(pressed) img_draw_dsc.recolor_opa = LV_OPA_30;

                lv_draw_image(base_dsc->layer, &img_draw_dsc, &a);

            }
        }
    }
}

/**
 * Add custom drawer to the button matrix to customize buttons one by one
 */
void lv_example_buttonmatrix_2(void)
{
    lv_obj_t * btnm = lv_buttonmatrix_create(lv_screen_active());
    lv_obj_add_event_cb(btnm, event_cb, LV_EVENT_DRAW_TASK_ADDED, NULL);
    lv_obj_add_flag(btnm, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);
    lv_obj_center(btnm);
}

#endif
```

Paginazione

```
#include "../../lv_examples.h"
#if LV_USE_BUTTONMATRIX  && LV_BUILD_EXAMPLES

static void event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    uint32_t id = lv_buttonmatrix_get_selected_button(obj);
    bool prev = id == 0;
    bool next = id == 6;
    if(prev || next) {
        /*Find the checked button*/
        uint32_t i;
        for(i = 1; i < 7; i++) {
            if(lv_buttonmatrix_has_button_ctrl(obj, i, LV_BUTTONMATRIX_
↪CTRL_CHECKED)) break;
        }

        if(prev && i > 1) i--;
        else if(next && i < 5) i++;
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_buttonmatrix_set_button_ctrl(obj, i, LV_BUTTONMATRIX_CTRL_
↪CHECKED);
    }
}

/**
 * Make a button group (pagination)
 */
void lv_example_buttonmatrix_3(void)
{
    static lv_style_t style_bg;
    lv_style_init(&style_bg);
    lv_style_set_pad_all(&style_bg, 0);
    lv_style_set_pad_gap(&style_bg, 0);
    lv_style_set_clip_corner(&style_bg, true);
    lv_style_set_radius(&style_bg, LV_RADIUS_CIRCLE);
    lv_style_set_border_width(&style_bg, 0);

    static lv_style_t style_btn;
    lv_style_init(&style_btn);
    lv_style_set_radius(&style_btn, 0);
    lv_style_set_border_width(&style_btn, 1);
    lv_style_set_border_opa(&style_btn, LV_OPA_50);
    lv_style_set_border_color(&style_btn, lv_palette_main(LV_PALETTE_
↪GREY));
    lv_style_set_border_side(&style_btn, LV_BORDER_SIDE_INTERNAL);
    lv_style_set_radius(&style_btn, 0);

    static const char * map[] = {LV_SYMBOL_LEFT, "1", "2", "3", "4", "5",␣
↪LV_SYMBOL_RIGHT, ""};

    lv_obj_t * btnm = lv_buttonmatrix_create(lv_screen_active());
    lv_buttonmatrix_set_map(btnm, map);
    lv_obj_add_style(btnm, &style_bg, 0);
    lv_obj_add_style(btnm, &style_btn, LV_PART_ITEMS);
    lv_obj_add_event_cb(btnm, event_cb, LV_EVENT_VALUE_CHANGED, NULL);
    lv_obj_set_size(btnm, 225, 35);

    /*Allow selecting on one number at time*/
    lv_buttonmatrix_set_button_ctrl_all(btnm, LV_BUTTONMATRIX_CTRL_
↪CHECKABLE);
    lv_buttonmatrix_clear_button_ctrl(btnm, 0, LV_BUTTONMATRIX_CTRL_
↪CHECKABLE);
    lv_buttonmatrix_clear_button_ctrl(btnm, 6, LV_BUTTONMATRIX_CTRL_
↪CHECKABLE);

    lv_buttonmatrix_set_one_checked(btnm, true);
    lv_buttonmatrix_set_button_ctrl(btnm, 1, LV_BUTTONMATRIX_CTRL_CHECKED);

    lv_obj_center(btnm);

}

#endif
```

3.7.7 Calendario

Calendario con intestazione

```
#include "../../lv_examples.h"
#if LV_USE_CALENDAR && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = (lv_obj_t *)lv_event_get_current_target(e);

    if(code == LV_EVENT_VALUE_CHANGED) {
        lv_calendar_date_t date;
        if(lv_calendar_get_pressed_date(obj, &date)) {
            LV_LOG_USER("Clicked date: %02d.%02d.%d", date.day, date.month,
↪ date.year);
        }
    }
}

void lv_example_calendar_1(void)
{
    lv_obj_t  * calendar = lv_calendar_create(lv_screen_active());
    lv_obj_set_size(calendar, 185, 230);
    lv_obj_align(calendar, LV_ALIGN_CENTER, 0, 27);
    lv_obj_add_event_cb(calendar, event_handler, LV_EVENT_ALL, NULL);

    lv_calendar_set_today_date(calendar, 2021, 02, 23);
    lv_calendar_set_month_shown(calendar, 2021, 02);

    /*Highlight a few days*/
    static lv_calendar_date_t highlighted_days[3];       /*Only its␣
↪pointer will be saved so should be static*/
    highlighted_days[0].year = 2021;
    highlighted_days[0].month = 02;
    highlighted_days[0].day = 6;

    highlighted_days[1].year = 2021;
    highlighted_days[1].month = 02;
    highlighted_days[1].day = 11;

    highlighted_days[2].year = 2022;
    highlighted_days[2].month = 02;
    highlighted_days[2].day = 22;

    lv_calendar_set_highlighted_dates(calendar, highlighted_days, 3);

#if LV_USE_CALENDAR_HEADER_DROPDOWN
    lv_calendar_add_header_dropdown(calendar);
#elif LV_USE_CALENDAR_HEADER_ARROW
    lv_calendar_add_header_arrow(calendar);
#endif
}

#endif
```

Calendario cinese

```
#include "../../lv_examples.h"
#if LV_USE_CALENDAR && LV_USE_CALENDAR_CHINESE && LV_BUILD_EXAMPLES

void lv_example_calendar_2(void)
{
    lv_obj_t  * calendar = lv_calendar_create(lv_screen_active());
    lv_obj_set_size(calendar, 300, 300);
    lv_obj_align(calendar, LV_ALIGN_TOP_MID, 0, 0);

    lv_calendar_set_today_date(calendar, 2024, 03, 22);
    lv_calendar_set_month_shown(calendar, 2024, 03);

#if LV_USE_CALENDAR_HEADER_DROPDOWN
    lv_calendar_add_header_dropdown(calendar);
#elif LV_USE_CALENDAR_HEADER_ARROW
    lv_calendar_add_header_arrow(calendar);
#endif

    lv_calendar_set_chinese_mode(calendar, true);
    lv_obj_set_style_text_font(calendar, &lv_font_source_han_sans_sc_14_
↪cjk, LV_PART_MAIN);
}

#else

void lv_example_calendar_2(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "chinese calendar is not enabled");
    lv_obj_center(label);
}

#endif
```

3.7.8 Canvas

Disegno sulla Canvas e rotazione

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  200
#define CANVAS_HEIGHT  150

void lv_example_canvas_1(void)
{
    lv_draw_rect_dsc_t rect_dsc;
    lv_draw_rect_dsc_init(&rect_dsc);
    rect_dsc.radius = 10;
    rect_dsc.bg_opa = LV_OPA_COVER;
    rect_dsc.bg_grad.dir = LV_GRAD_DIR_VER;
    rect_dsc.bg_grad.stops[0].color = lv_palette_main(LV_PALETTE_RED);
    rect_dsc.bg_grad.stops[0].opa = LV_OPA_100;
    rect_dsc.bg_grad.stops[1].color = lv_palette_main(LV_PALETTE_BLUE);
    rect_dsc.bg_grad.stops[1].opa = LV_OPA_50;
```

(continues on next page)

(continua dalla pagina precedente)

```
    rect_dsc.border_width = 2;
    rect_dsc.border_opa = LV_OPA_90;
    rect_dsc.border_color = lv_color_white();
    rect_dsc.shadow_width = 5;
    rect_dsc.shadow_offset_x = 5;
    rect_dsc.shadow_offset_y = 5;

    lv_draw_label_dsc_t label_dsc;
    lv_draw_label_dsc_init(&label_dsc);
    label_dsc.color = lv_palette_main(LV_PALETTE_ORANGE);
    label_dsc.text = "Some text on text canvas";
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf_16bpp, CANVAS_WIDTH, CANVAS_HEIGHT,␣
↪LV_COLOR_FORMAT_RGB565);
    LV_DRAW_BUF_INIT_STATIC(draw_buf_16bpp);

    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf_16bpp);
    lv_obj_center(canvas);
    lv_canvas_fill_bg(canvas, lv_palette_lighten(LV_PALETTE_GREY, 3), LV_
↪OPA_COVER);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_area_t coords_rect = {30, 20, 100, 70};
    lv_draw_rect(&layer, &rect_dsc, &coords_rect);

    lv_area_t coords_text = {40, 80, 100, 120};
    lv_draw_label(&layer, &label_dsc, &coords_text);

    lv_canvas_finish_layer(canvas, &layer);

    /*Test the rotation. It requires another buffer where the original␣
↪image is stored.
     *So use previous canvas as image and rotate it to the new canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf_32bpp, CANVAS_WIDTH, CANVAS_HEIGHT,␣
↪LV_COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf_32bpp);

    /*Create a canvas and initialize its palette*/
    canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf_32bpp);
    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_canvas_fill_bg(canvas, lv_palette_lighten(LV_PALETTE_GREY, 1), LV_
↪OPA_COVER);

    lv_canvas_init_layer(canvas, &layer);
    lv_image_dsc_t img;
    lv_draw_buf_to_image(&draw_buf_16bpp, &img);
    lv_draw_image_dsc_t img_dsc;
    lv_draw_image_dsc_init(&img_dsc);
    img_dsc.rotation = 120;
    img_dsc.src = &img;
```

(continues on next page)

(continua dalla pagina precedente)

```
    img_dsc.pivot.x = CANVAS_WIDTH / 2;
    img_dsc.pivot.y = CANVAS_HEIGHT / 2;

    lv_area_t coords_img = {0, 0, CANVAS_WIDTH - 1, CANVAS_HEIGHT - 1};
    lv_draw_image(&layer, &img_dsc, &coords_img);

    lv_canvas_finish_layer(canvas, &layer);
}

#endif
```

Canvas trasparente con chroma key

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  80
#define CANVAS_HEIGHT  40

/**
 * Create a transparent canvas with transparency
 */
void lv_example_canvas_2(void)
{
    lv_obj_set_style_bg_color(lv_screen_active(), lv_palette_lighten(LV_
↪PALETTE_RED, 5), 0);

    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf);
    lv_obj_center(canvas);

    /*Red background (There is no dedicated alpha channel in indexed␣
↪images so LV_OPA_COVER is ignored)*/
    lv_canvas_fill_bg(canvas, lv_palette_main(LV_PALETTE_BLUE), LV_OPA_
↪COVER);

    /*Create hole on the canvas*/
    int32_t x;
    int32_t y;
    for(y = 10; y < 20; y++) {
        for(x = 5; x < 75; x++) {
            lv_canvas_set_px(canvas, x, y, lv_palette_main(LV_PALETTE_
↪BLUE), LV_OPA_50);
        }
    }

    for(y = 20; y < 30; y++) {
        for(x = 5; x < 75; x++) {
            lv_canvas_set_px(canvas, x, y, lv_palette_main(LV_PALETTE_
↪BLUE), LV_OPA_20);
```

(continues on next page)

(continua dalla pagina precedente)

```
        }
    }

    for(y = 30; y < 40; y++) {
        for(x = 5; x < 75; x++) {
            lv_canvas_set_px(canvas, x, y, lv_palette_main(LV_PALETTE_
↪BLUE), LV_OPA_0);
        }
    }
}
#endif
```

Disegnare un rettangolo sul canvas

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  50
#define CANVAS_HEIGHT  50

/**
 * Draw a rectangle to the canvas
 */
void lv_example_canvas_3(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf);

    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_draw_rect_dsc_t dsc;
    lv_draw_rect_dsc_init(&dsc);
    dsc.bg_color = lv_palette_main(LV_PALETTE_RED);
    dsc.border_color = lv_palette_main(LV_PALETTE_BLUE);
    dsc.border_width = 3;
    dsc.outline_color = lv_palette_main(LV_PALETTE_GREEN);
    dsc.outline_width = 2;
    dsc.outline_pad = 2;
    dsc.outline_opa = LV_OPA_50;
    dsc.radius = 5;
    dsc.border_width = 3;

    lv_area_t coords = {10, 10, 40, 30};

    lv_draw_rect(&layer, &dsc, &coords);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_canvas_finish_layer(canvas, &layer);
}
#endif
```

Disegnare un'etichetta sul canvas

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_FONT_MONTSERRAT_18 && LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  50
#define CANVAS_HEIGHT  50

/**
 * Draw a text to the canvas
 */
void lv_example_canvas_4(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf);
    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_draw_label_dsc_t dsc;
    lv_draw_label_dsc_init(&dsc);
    dsc.color = lv_palette_main(LV_PALETTE_RED);
    dsc.font = &lv_font_montserrat_18;
    dsc.decor = LV_TEXT_DECOR_UNDERLINE;
    dsc.text = "Hello";

    lv_area_t coords = {10, 10, 30, 60};

    lv_draw_label(&layer, &dsc, &coords);

    lv_canvas_finish_layer(canvas, &layer);

}
#endif
```

Disegnare un arco sul canvas

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  50
#define CANVAS_HEIGHT  50
```

(continues on next page)

(continua dalla pagina precedente)

```
/**
 * Draw an arc to the canvas
 */
void lv_example_canvas_5(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf);
    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_draw_arc_dsc_t dsc;
    lv_draw_arc_dsc_init(&dsc);
    dsc.color = lv_palette_main(LV_PALETTE_RED);
    dsc.width = 5;
    dsc.center.x = 25;
    dsc.center.y = 25;
    dsc.width = 10;
    dsc.radius = 15;
    dsc.start_angle = 0;
    dsc.end_angle = 220;

    lv_draw_arc(&layer, &dsc);

    lv_canvas_finish_layer(canvas, &layer);

}
#endif
```

Disegnare un'immagine sul canvas

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  50
#define CANVAS_HEIGHT  50

/**
 * Draw an image to the canvas
 */
void lv_example_canvas_6(void)
{
    /*Create a buffer for the canvas*/
    static uint8_t cbuf[LV_CANVAS_BUF_SIZE(CANVAS_WIDTH, CANVAS_HEIGHT, 32,
↪ LV_DRAW_BUF_STRIDE_ALIGN)];

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_canvas_set_buffer(canvas, cbuf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    LV_IMAGE_DECLARE(img_star);
    lv_draw_image_dsc_t dsc;
    lv_draw_image_dsc_init(&dsc);
    dsc.src = &img_star;

    lv_area_t coords = {10, 10, 10 + img_star.header.w - 1, 10 + img_star.
↪header.h - 1};

    lv_draw_image(&layer, &dsc, &coords);

    lv_canvas_finish_layer(canvas, &layer);
}
#endif
```

Disegnare una linea sul canvas

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS&& LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  50
#define CANVAS_HEIGHT  50

/**
 * Draw a line to the canvas
 */
void lv_example_canvas_7(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf);
    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_draw_line_dsc_t dsc;
    lv_draw_line_dsc_init(&dsc);
    dsc.color = lv_palette_main(LV_PALETTE_RED);
    dsc.width = 4;
    dsc.round_end = 1;
    dsc.round_start = 1;
    dsc.p1.x = 15;
```

(continues on next page)

(continua dalla pagina precedente)

```
    dsc.p1.y = 15;
    dsc.p2.x = 35;
    dsc.p2.y = 10;
    lv_draw_line(&layer, &dsc);

    lv_canvas_finish_layer(canvas, &layer);

}
#endif
```

Disegnare un'immagine vettoriale sul canvas

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES

#if LV_USE_VECTOR_GRAPHIC

#define CANVAS_WIDTH  150
#define CANVAS_HEIGHT 150

/**
 * Draw a path to the canvas
 */
void lv_example_canvas_8(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf);
    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_vector_dsc_t * dsc = lv_vector_dsc_create(&layer);
    lv_vector_path_t * path = lv_vector_path_create(LV_VECTOR_PATH_QUALITY_
↪MEDIUM);

    lv_fpoint_t pts[] = {{10, 10}, {130, 130}, {10, 130}};
    lv_vector_path_move_to(path, &pts[0]);
    lv_vector_path_line_to(path, &pts[1]);
    lv_vector_path_line_to(path, &pts[2]);
    lv_vector_path_close(path);

    lv_vector_dsc_set_fill_color(dsc, lv_color_make(0x00, 0x80, 0xff));
    lv_vector_dsc_add_path(dsc, path);

    lv_draw_vector(dsc);
    lv_vector_path_delete(path);
    lv_vector_dsc_delete(dsc);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_canvas_finish_layer(canvas, &layer);
}
#else

void lv_example_canvas_8(void)
{
    /*fallback for online examples*/
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Vector graphics is not enabled");
    lv_obj_center(label);
}

#endif /*LV_USE_VECTOR_GRAPHIC*/

#endif
```

Disegnare un triangolo sul canvas

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES


#define CANVAS_WIDTH  150
#define CANVAS_HEIGHT 150

/**
 * Draw a triangle to the canvas
 */
void lv_example_canvas_9(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf);
    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_draw_triangle_dsc_t tri_dsc;
    lv_draw_triangle_dsc_init(&tri_dsc);
    tri_dsc.p[0].x = 10;
    tri_dsc.p[0].y = 10;
    tri_dsc.p[1].x = 100;
    tri_dsc.p[1].y = 30;
    tri_dsc.p[2].x = 50;
    tri_dsc.p[2].y = 100;

    tri_dsc.grad.stops_count = 2;
    tri_dsc.grad.dir = LV_GRAD_DIR_VER;
    tri_dsc.grad.stops[0].color = lv_color_hex(0xff0000);
```

(continues on next page)

(continua dalla pagina precedente)

```
    tri_dsc.grad.stops[0].frac = 64;    /*Start at 25%*/
    tri_dsc.grad.stops[0].opa = LV_OPA_COVER;
    tri_dsc.grad.stops[1].color = lv_color_hex(0x0000ff);
    tri_dsc.grad.stops[1].opa = LV_OPA_TRANSP;
    tri_dsc.grad.stops[1].frac = 3 * 64;    /*End at 75%*/

    tri_dsc.opa = 128;  /*Set the overall opacity to 50%*/

    lv_draw_triangle(&layer, &tri_dsc);

    lv_canvas_finish_layer(canvas, &layer);
}

#endif
```

Disegnare Effetti di Lettere Fantasiose

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  300
#define CANVAS_HEIGHT  200

static void timer_cb(lv_timer_t * timer)
{
    static int16_t counter = 0;
    const char * string = "lol~ I'm wavvvvvvving~>>>";
    const int16_t string_len = lv_strlen(string);

    lv_obj_t * canvas = (lv_obj_t *)lv_timer_get_user_data(timer);
    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_canvas_fill_bg(canvas, lv_color_white(), LV_OPA_COVER);

    lv_draw_letter_dsc_t letter_dsc;
    lv_draw_letter_dsc_init(&letter_dsc);
    letter_dsc.color = lv_color_hex(0xff0000);
    letter_dsc.font = lv_font_get_default();

    {
#define CURVE2_X(t) (t * 2 + 10)
#define CURVE2_Y(t) (lv_trigo_sin((t) * 5) * 40 / 32767 + CANVAS_HEIGHT /␣
↪2)

        int32_t pre_x = CURVE2_X(-1);
        int32_t pre_y = CURVE2_Y(-1);
        for(int16_t i = 0; i < string_len; i++) {
            const int16_t angle = (int16_t)(i * 5);
            const int32_t x = CURVE2_X(angle);
            const int32_t y = CURVE2_Y(angle + counter / 2);
            const lv_point_t point = { .x = x, .y = y };

            letter_dsc.unicode = (uint32_t)string[i % string_len];
            letter_dsc.rotation = lv_atan2(y - pre_y, x - pre_x) * 10;
            letter_dsc.color = lv_color_hsv_to_rgb(i * 10, 100, 100);
```

(continues on next page)

(continua dalla pagina precedente)

```
            lv_draw_letter(&layer, &letter_dsc, &point);

            pre_x = x;
            pre_y = y;
        }
    }

    lv_canvas_finish_layer(canvas, &layer);

    counter++;
}

void lv_example_canvas_10(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_obj_set_size(canvas, CANVAS_WIDTH, CANVAS_HEIGHT);

    lv_obj_center(canvas);

    lv_canvas_set_draw_buf(canvas, &draw_buf);

    lv_timer_create(timer_cb, 16, canvas);
}

#endif
```

Disegnare Effetti di Lettere Fantasiose 2

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  300
#define CANVAS_HEIGHT  200

static void timer_cb(lv_timer_t * timer)
{
    static int32_t counter = 0;
    const char * string = "windstorrrrrrrrrrrrrrrm~>>>";
    const int16_t string_len = lv_strlen(string);

    lv_obj_t * canvas = (lv_obj_t *) lv_timer_get_user_data(timer);
    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_canvas_fill_bg(canvas, lv_color_white(), LV_OPA_COVER);

    lv_draw_letter_dsc_t letter_dsc;
    lv_draw_letter_dsc_init(&letter_dsc);
    letter_dsc.color = lv_color_hex(0xff0000);
    letter_dsc.font = lv_font_get_default();
```

(continues on next page)

(continua dalla pagina precedente)

```
    {
#define CURVE2_X(t) ((t) * 2 + lv_trigo_cos((t) * 5) * 40 / 32767 - 10)
#define CURVE2_Y(t, T) ((t) * lv_trigo_sin(((t) + (T)) * 5) * 40 / 32767 /␣
↪80 + CANVAS_HEIGHT / 2)

        int32_t pre_x = CURVE2_X(-1);
        int32_t pre_y = CURVE2_Y(-1, 0);
        for(int16_t i = 0; i < string_len; i++) {
            const int32_t angle = i * 5;
            const int32_t x = CURVE2_X(angle);
            const int32_t y = CURVE2_Y(angle + 30, counter / 2);

            letter_dsc.unicode = (uint32_t)string[i % string_len];
            letter_dsc.rotation = lv_atan2(y - pre_y, x - pre_x) * 10;
            letter_dsc.color = lv_color_hsv_to_rgb(i * 10, 100, 100);
            lv_point_t p = (lv_point_t) {
                .x = x, .y = y
            };
            lv_draw_letter(&layer, &letter_dsc, &p);

            pre_x = x;
            pre_y = y;
        }
    }

    lv_canvas_finish_layer(canvas, &layer);

    counter++;
}

void lv_example_canvas_11(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_obj_set_size(canvas, CANVAS_WIDTH, CANVAS_HEIGHT);

    lv_obj_center(canvas);

    lv_canvas_set_draw_buf(canvas, &draw_buf);

    lv_timer_create(timer_cb, 16, canvas);
}

#endif
```

3.7.9 Grafico

Grafico a linee

```
#include "../../lv_examples.h"
#if LV_USE_CHART && LV_BUILD_EXAMPLES
```

(continues on next page)

(continua dalla pagina precedente)

```
/**
 * A very basic line chart
 */
void lv_example_chart_1(void)
{
    /*Create a chart*/
    lv_obj_t * chart;
    chart = lv_chart_create(lv_screen_active());
    lv_obj_set_size(chart, 200, 150);
    lv_obj_center(chart);
    lv_chart_set_type(chart, LV_CHART_TYPE_LINE);   /*Show lines and␣
↪points too*/

    /*Add two data series*/
    lv_chart_series_t * ser1 = lv_chart_add_series(chart, lv_palette_
↪main(LV_PALETTE_GREEN), LV_CHART_AXIS_PRIMARY_Y);
    lv_chart_series_t * ser2 = lv_chart_add_series(chart, lv_palette_
↪main(LV_PALETTE_RED), LV_CHART_AXIS_SECONDARY_Y);
    int32_t * ser2_y_points = lv_chart_get_series_y_array(chart, ser2);

    uint32_t i;
    for(i = 0; i < 10; i++) {
        /*Set the next points on 'ser1'*/
        lv_chart_set_next_value(chart, ser1, (int32_t)lv_rand(10, 50));

        /*Directly set points on 'ser2'*/
        ser2_y_points[i] = (int32_t)lv_rand(50, 90);
    }

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

#endif
```

Segni di spunta e etichette degli assi con scorrimento su un grafico

```
#include "../../lv_examples.h"
#if LV_USE_CHART && LV_BUILD_EXAMPLES

/**
 * Use lv_scale to add ticks to a scrollable chart
 */
void lv_example_chart_2(void)
{
    /*Create a container*/
    lv_obj_t * main_cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(main_cont, 200, 150);
    lv_obj_center(main_cont);

    /*Create a transparent wrapper for the chart and the scale.
     *Set a large width, to make it scrollable on the main container*/
    lv_obj_t * wrapper = lv_obj_create(main_cont);
    lv_obj_remove_style_all(wrapper);
    lv_obj_set_size(wrapper, lv_pct(300), lv_pct(100));
    lv_obj_set_flex_flow(wrapper, LV_FLEX_FLOW_COLUMN);
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Create a chart on the wrapper
     *Set it's width to 100% to fill the large wrapper*/
    lv_obj_t * chart = lv_chart_create(wrapper);
    lv_obj_set_width(chart, lv_pct(100));
    lv_obj_set_flex_grow(chart, 1);
    lv_chart_set_type(chart, LV_CHART_TYPE_BAR);
    lv_chart_set_axis_range(chart, LV_CHART_AXIS_PRIMARY_Y, 0, 100);
    lv_chart_set_axis_range(chart, LV_CHART_AXIS_SECONDARY_Y, 0, 400);
    lv_chart_set_point_count(chart, 12);
    lv_obj_set_style_radius(chart, 0, 0);

    /*Create a scale also with 100% width*/
    lv_obj_t * scale_bottom = lv_scale_create(wrapper);
    lv_scale_set_mode(scale_bottom, LV_SCALE_MODE_HORIZONTAL_BOTTOM);
    lv_obj_set_size(scale_bottom, lv_pct(100), 25);
    lv_scale_set_total_tick_count(scale_bottom, 12);
    lv_scale_set_major_tick_every(scale_bottom, 1);
    lv_obj_set_style_pad_hor(scale_bottom, lv_chart_get_first_point_center_
↪offset(chart), 0);

    static const char * month[] = {"Jan", "Febr", "March", "Apr", "May",
↪"Jun", "July", "Aug", "Sept", "Oct", "Nov", "Dec", NULL};
    lv_scale_set_text_src(scale_bottom, month);

    /*Add two data series*/
    lv_chart_series_t * ser1 = lv_chart_add_series(chart, lv_palette_
↪lighten(LV_PALETTE_GREEN, 2), LV_CHART_AXIS_PRIMARY_Y);
    lv_chart_series_t * ser2 = lv_chart_add_series(chart, lv_palette_
↪darken(LV_PALETTE_GREEN, 2), LV_CHART_AXIS_PRIMARY_Y);

    /*Set the next points on 'ser1'*/
    uint32_t i;
    for(i = 0; i < 12; i++) {
        lv_chart_set_next_value(chart, ser1, (int32_t)lv_rand(10, 60));
        lv_chart_set_next_value(chart, ser2, (int32_t)lv_rand(50, 90));
    }
    lv_chart_refresh(chart); /*Required after direct set*/
}

#endif
```

Mostrare il valore dei punti premuti

```
#include "../../lv_examples.h"
#if LV_USE_CHART && LV_BUILD_EXAMPLES

static void event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * chart = lv_event_get_target_obj(e);

    if(code == LV_EVENT_VALUE_CHANGED) {
        lv_obj_invalidate(chart);
    }
    if(code == LV_EVENT_REFR_EXT_DRAW_SIZE) {
        int32_t * s = (int32_t *)lv_event_get_param(e);
```

(continues on next page)

(continua dalla pagina precedente)

```
        *s = LV_MAX(*s, 20);
    }
    else if(code == LV_EVENT_DRAW_POST_END) {
        uint32_t id = lv_chart_get_pressed_point(chart);
        if(id == LV_CHART_POINT_NONE) return;

        LV_LOG_USER("Selected point %d", (int)id);

        lv_chart_series_t * ser = lv_chart_get_series_next(chart, NULL);
        while(ser) {
            lv_point_t p;
            lv_chart_get_point_pos_by_id(chart, ser, id, &p);

            int32_t * y_array = lv_chart_get_series_y_array(chart, ser);
            int32_t value = y_array[id];

            /*Draw a rectangle above the clicked point*/
            lv_layer_t * layer = lv_event_get_layer(e);
            lv_draw_rect_dsc_t draw_rect_dsc;
            lv_draw_rect_dsc_init(&draw_rect_dsc);
            draw_rect_dsc.bg_color = lv_color_black();
            draw_rect_dsc.bg_opa = LV_OPA_50;
            draw_rect_dsc.radius = 3;

            lv_area_t chart_obj_coords;
            lv_obj_get_coords(chart, &chart_obj_coords);
            lv_area_t rect_area;
            rect_area.x1 = chart_obj_coords.x1 + p.x - 20;
            rect_area.x2 = chart_obj_coords.x1 + p.x + 20;
            rect_area.y1 = chart_obj_coords.y1 + p.y - 30;
            rect_area.y2 = chart_obj_coords.y1 + p.y - 10;
            lv_draw_rect(layer, &draw_rect_dsc, &rect_area);

            /*Draw the value as label to the center of the rectangle*/
            char buf[16];
            lv_snprintf(buf, sizeof(buf), LV_SYMBOL_DUMMY"$%d", value);

            lv_draw_label_dsc_t draw_label_dsc;
            lv_draw_label_dsc_init(&draw_label_dsc);
            draw_label_dsc.color = lv_color_white();
            draw_label_dsc.text = buf;
            draw_label_dsc.text_local = 1;
            draw_label_dsc.align = LV_TEXT_ALIGN_CENTER;
            lv_area_t label_area = rect_area;
            lv_area_set_height(&label_area, lv_font_get_line_height(draw_
↪label_dsc.font));
            lv_area_align(&rect_area, &label_area, LV_ALIGN_CENTER, 0, 0);
            lv_draw_label(layer, &draw_label_dsc, &label_area);

            ser = lv_chart_get_series_next(chart, ser);
        }
    }
    else if(code == LV_EVENT_RELEASED) {
        lv_obj_invalidate(chart);
    }
}
```

(continues on next page)

(continua dalla pagina precedente)

```
/**
 * Show the value of the pressed points
 */
void lv_example_chart_3(void)
{
    /*Create a chart*/
    lv_obj_t * chart;
    chart = lv_chart_create(lv_screen_active());
    lv_obj_set_size(chart, 200, 150);
    lv_obj_center(chart);

    lv_obj_add_event_cb(chart, event_cb, LV_EVENT_ALL, NULL);
    lv_obj_refresh_ext_draw_size(chart);

    /*Zoom in a little in X*/
    //    lv_chart_set_scale_x(chart, 800);

    /*Add two data series*/
    lv_chart_series_t * ser1 = lv_chart_add_series(chart, lv_palette_
↪main(LV_PALETTE_RED), LV_CHART_AXIS_PRIMARY_Y);
    lv_chart_series_t * ser2 = lv_chart_add_series(chart, lv_palette_
↪main(LV_PALETTE_GREEN), LV_CHART_AXIS_PRIMARY_Y);
    uint32_t i;
    for(i = 0; i < 10; i++) {
        lv_chart_set_next_value(chart, ser1, (int32_t)lv_rand(60, 90));
        lv_chart_set_next_value(chart, ser2, (int32_t)lv_rand(10, 40));
    }
}

#endif
```

Ricolorare le barre in base al loro valore

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

#if LV_USE_CHART && LV_DRAW_SW_COMPLEX && LV_BUILD_EXAMPLES

static void draw_event_cb(lv_event_t * e)
{
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);

    if(base_dsc->part != LV_PART_ITEMS) {
        return;
    }

    lv_draw_fill_dsc_t * fill_dsc = lv_draw_task_get_fill_dsc(draw_task);
    if(fill_dsc) {
        lv_obj_t * chart = lv_event_get_target_obj(e);
        int32_t * y_array = lv_chart_get_series_y_array(chart, lv_chart_
↪get_series_next(chart, NULL));
        int32_t v = y_array[base_dsc->id2];

        uint8_t ratio = (uint8_t)(v * 255 / 100);
```

(continues on next page)

(continua dalla pagina precedente)

```
        fill_dsc->color = lv_color_mix(lv_palette_main(LV_PALETTE_GREEN),␣
↪lv_palette_main(LV_PALETTE_RED), ratio);
    }
}

/**
 * Recolor the bars of a chart based on their value
 */
void lv_example_chart_4(void)
{
    /*Create a chart1*/
    lv_obj_t * chart = lv_chart_create(lv_screen_active());
    lv_chart_set_type(chart, LV_CHART_TYPE_BAR);
    lv_chart_set_point_count(chart, 24);
    lv_obj_set_style_pad_column(chart, 2, 0);
    lv_obj_set_size(chart, 260, 160);
    lv_obj_center(chart);

    lv_chart_series_t * ser = lv_chart_add_series(chart, lv_color_
↪hex(0xff0000), LV_CHART_AXIS_PRIMARY_Y);
    lv_obj_add_event_cb(chart, draw_event_cb, LV_EVENT_DRAW_TASK_ADDED,␣
↪NULL);
    lv_obj_add_flag(chart, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);

    uint32_t i;
    for(i = 0; i < 24; i++) {
        lv_chart_set_next_value(chart, ser, (int32_t)lv_rand(10, 90));
    }
}

#endif
```

Grafico a linee sfumate con linee di divisione personalizzate

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

#if LV_USE_CHART && LV_DRAW_SW_COMPLEX && LV_BUILD_EXAMPLES
#include "../../../lvgl_private.h"

static void hook_division_lines(lv_event_t * e);
static void add_faded_area(lv_event_t * e);
static void draw_event_cb(lv_event_t * e);

/**
 * Add a faded area effect to the line chart and make some division lines␣
↪ticker
 */
void lv_example_chart_5(void)
{
    /*Create a chart*/
    lv_obj_t * chart = lv_chart_create(lv_screen_active());
    lv_chart_set_type(chart, LV_CHART_TYPE_LINE);   /*Show lines and␣
↪points too*/
    lv_obj_set_size(chart, 200, 150);
    lv_obj_set_style_pad_all(chart, 0, 0);
    lv_obj_set_style_radius(chart, 0, 0);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_center(chart);

    lv_chart_set_div_line_count(chart, 5, 7);

    lv_obj_add_event_cb(chart, draw_event_cb, LV_EVENT_DRAW_TASK_ADDED,␣
↪NULL);
    lv_obj_add_flag(chart, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);

    lv_chart_series_t * ser = lv_chart_add_series(chart, lv_palette_
↪main(LV_PALETTE_RED), LV_CHART_AXIS_PRIMARY_Y);

    uint32_t i;
    for(i = 0; i < 10; i++) {
        lv_chart_set_next_value(chart, ser, (int32_t)lv_rand(10, 80));
    }
}

static void draw_event_cb(lv_event_t * e)
{
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);

    if(base_dsc->part == LV_PART_ITEMS && lv_draw_task_get_type(draw_task)␣
↪== LV_DRAW_TASK_TYPE_LINE) {
        add_faded_area(e);

    }
    /*Hook the division lines too*/
    if(base_dsc->part == LV_PART_MAIN && lv_draw_task_get_type(draw_task)␣
↪== LV_DRAW_TASK_TYPE_LINE) {
        hook_division_lines(e);
    }
}

static void add_faded_area(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    lv_area_t coords;
    lv_obj_get_coords(obj, &coords);

    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);

    const lv_chart_series_t * ser = lv_chart_get_series_next(obj, NULL);
    lv_color_t ser_color = lv_chart_get_series_color(obj, ser);

    /*Draw a triangle below the line witch some opacity gradient*/
    lv_draw_line_dsc_t * draw_line_dsc = (lv_draw_line_dsc_t *)lv_draw_
↪task_get_draw_dsc(draw_task);
    lv_draw_triangle_dsc_t tri_dsc;

    lv_draw_triangle_dsc_init(&tri_dsc);
    tri_dsc.p[0].x = draw_line_dsc->p1.x;
    tri_dsc.p[0].y = draw_line_dsc->p1.y;
```

(continues on next page)

(continua dalla pagina precedente)

```
    tri_dsc.p[1].x = draw_line_dsc->p2.x;
    tri_dsc.p[1].y = draw_line_dsc->p2.y;
    tri_dsc.p[2].x = draw_line_dsc->p1.y < draw_line_dsc->p2.y ? draw_line_
↪dsc->p1.x : draw_line_dsc->p2.x;
    tri_dsc.p[2].y = LV_MAX(draw_line_dsc->p1.y, draw_line_dsc->p2.y);
    tri_dsc.grad.dir = LV_GRAD_DIR_VER;

    int32_t full_h = lv_obj_get_height(obj);
    int32_t fract_uppter = (int32_t)(LV_MIN(draw_line_dsc->p1.y, draw_line_
↪dsc->p2.y) - coords.y1) * 255 / full_h;
    int32_t fract_lower = (int32_t)(LV_MAX(draw_line_dsc->p1.y, draw_line_
↪dsc->p2.y) - coords.y1) * 255 / full_h;
    tri_dsc.grad.stops[0].color = ser_color;
    tri_dsc.grad.stops[0].opa = (lv_opa_t)(255 - fract_uppter);
    tri_dsc.grad.stops[0].opa = 255 - fract_uppter;
    tri_dsc.grad.stops[0].frac = 0;
    tri_dsc.grad.stops[1].color = ser_color;
    tri_dsc.grad.stops[1].opa = (lv_opa_t)(255 - fract_lower);
    tri_dsc.grad.stops[1].frac = 255;

    lv_draw_triangle(base_dsc->layer, &tri_dsc);

    /*Draw rectangle below the triangle*/
    lv_draw_rect_dsc_t rect_dsc;
    lv_draw_rect_dsc_init(&rect_dsc);
    rect_dsc.bg_grad.dir = LV_GRAD_DIR_VER;
    rect_dsc.bg_grad.stops[0].color = ser_color;
    rect_dsc.bg_grad.stops[0].frac = 0;
    rect_dsc.bg_grad.stops[0].opa = (lv_opa_t)(255 - fract_lower);
    rect_dsc.bg_grad.stops[1].color = ser_color;
    rect_dsc.bg_grad.stops[1].frac = 255;
    rect_dsc.bg_grad.stops[1].opa = 0;

    lv_area_t rect_area;
    rect_area.x1 = (int32_t)draw_line_dsc->p1.x;
    rect_area.x2 = (int32_t)draw_line_dsc->p2.x - 1;
    rect_area.y1 = (int32_t)LV_MAX(draw_line_dsc->p1.y, draw_line_dsc->p2.
↪y) - 1;
    rect_area.y2 = (int32_t)coords.y2;
    lv_draw_rect(base_dsc->layer, &rect_dsc, &rect_area);
}

static void hook_division_lines(lv_event_t * e)
{
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);
    lv_draw_line_dsc_t * line_dsc = (lv_draw_line_dsc_t *)lv_draw_task_get_
↪draw_dsc(draw_task);

    /*Vertical line*/
    if(line_dsc->p1.x == line_dsc->p2.x) {
        line_dsc->color  = lv_palette_lighten(LV_PALETTE_GREY, 1);
        if(base_dsc->id1 == 3) {
            line_dsc->width = 2;
            line_dsc->dash_gap = 0;
```

(continues on next page)

(continua dalla pagina precedente)

```
            line_dsc->dash_width = 0;
        }
        else {
            line_dsc->width = 1;
            line_dsc->dash_gap = 6;
            line_dsc->dash_width = 6;
        }
    }
    /*Horizontal line*/
    else {
        if(base_dsc->id1 == 2) {
            line_dsc->width  = 2;
            line_dsc->dash_gap  = 0;
            line_dsc->dash_width  = 0;
        }
        else {
            line_dsc->width = 2;
            line_dsc->dash_gap  = 6;
            line_dsc->dash_width  = 6;
        }

        if(base_dsc->id1 == 1  || base_dsc->id1 == 3) {
            line_dsc->color  = lv_palette_main(LV_PALETTE_GREEN);
        }
        else {
            line_dsc->color  = lv_palette_lighten(LV_PALETTE_GREY, 1);
        }
    }
}

#endif
```

Mostrare il cursore sul punto cliccato

```
#include "../../lv_examples.h"
#if LV_USE_CHART && LV_BUILD_EXAMPLES

static lv_obj_t * chart;
static lv_chart_series_t * ser;
static lv_chart_cursor_t * cursor;

static void value_changed_event_cb(lv_event_t * e)
{
    uint32_t last_id;
    lv_obj_t * obj = lv_event_get_target_obj(e);

    last_id = lv_chart_get_pressed_point(obj);
    if(last_id != LV_CHART_POINT_NONE) {
        lv_chart_set_cursor_point(obj, cursor, NULL, last_id);
    }
}

/**
 * Show cursor on the clicked point
 */
void lv_example_chart_6(void)
```

(continues on next page)

(continua dalla pagina precedente)

```
{
    chart = lv_chart_create(lv_screen_active());
    lv_obj_set_size(chart, 200, 150);
    lv_obj_align(chart, LV_ALIGN_CENTER, 0, -10);

    //    lv_chart_set_axis_tick(chart, LV_CHART_AXIS_PRIMARY_Y, 10, 5, 6,␣
↪5, true, 40);
    //    lv_chart_set_axis_tick(chart, LV_CHART_AXIS_PRIMARY_X, 10, 5, 10,
↪ 1, true, 30);

    lv_obj_add_event_cb(chart, value_changed_event_cb, LV_EVENT_VALUE_
↪CHANGED, NULL);
    lv_obj_refresh_ext_draw_size(chart);

    cursor = lv_chart_add_cursor(chart, lv_palette_main(LV_PALETTE_BLUE),␣
↪(lv_dir_t)(LV_DIR_LEFT | LV_DIR_BOTTOM));

    ser = lv_chart_add_series(chart, lv_palette_main(LV_PALETTE_RED), LV_
↪CHART_AXIS_PRIMARY_Y);
    uint32_t i;
    for(i = 0; i < 10; i++) {
        lv_chart_set_next_value(chart, ser, (int32_t)lv_rand(10, 90));
    }

    //    lv_chart_set_scale_x(chart, 500);

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Click on a point");
    lv_obj_align_to(label, chart, LV_ALIGN_OUT_TOP_MID, 0, -5);
}

#endif
```

Grafico a dispersione

```
#include "../../lv_examples.h"
#if LV_USE_CHART && LV_BUILD_EXAMPLES

static void draw_event_cb(lv_event_t * e)
{
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);
    if(base_dsc->part == LV_PART_INDICATOR) {
        lv_obj_t * obj = lv_event_get_target_obj(e);
        lv_chart_series_t * ser = lv_chart_get_series_next(obj, NULL);
        lv_draw_rect_dsc_t * rect_draw_dsc = (lv_draw_rect_dsc_t *)lv_draw_
↪task_get_draw_dsc(draw_task);
        uint32_t cnt = lv_chart_get_point_count(obj);

        /*Make older value more transparent*/
        rect_draw_dsc->bg_opa = (lv_opa_t)((LV_OPA_COVER * base_dsc->id2) /
↪ (cnt - 1));

        /*Make smaller values blue, higher values red*/
        int32_t * x_array = lv_chart_get_series_x_array(obj, ser);
```

(continues on next page)

(continua dalla pagina precedente)

```
        int32_t * y_array = lv_chart_get_series_y_array(obj, ser);
        /*dsc->id is the tells drawing order, but we need the ID of the␣
↪point being drawn.*/
        uint32_t start_point = lv_chart_get_x_start_point(obj, ser);
        uint32_t p_act = (start_point + base_dsc->id2) % cnt; /*Consider␣
↪start point to get the index of the array*/
        lv_opa_t x_opa = (lv_opa_t)((x_array[p_act] * LV_OPA_50) / 200);
        lv_opa_t y_opa = (lv_opa_t)((y_array[p_act] * LV_OPA_50) / 1000);

        rect_draw_dsc->bg_color = lv_color_mix(lv_palette_main(LV_PALETTE_
↪RED),
                                               lv_palette_main(LV_PALETTE_
↪BLUE),
                                               x_opa + y_opa);
    }
}

static void add_data(lv_timer_t * timer)
{
    lv_obj_t * chart = (lv_obj_t *)lv_timer_get_user_data(timer);
    lv_chart_set_next_value2(chart, lv_chart_get_series_next(chart, NULL),␣
↪(int32_t)lv_rand(0, 200),
                             (int32_t)lv_rand(0, 1000));
}

/**
 * A scatter chart
 */
void lv_example_chart_7(void)
{
    lv_obj_t * chart = lv_chart_create(lv_screen_active());
    lv_obj_set_size(chart, 200, 150);
    lv_obj_align(chart, LV_ALIGN_CENTER, 0, 0);
    lv_obj_add_event_cb(chart, draw_event_cb, LV_EVENT_DRAW_TASK_ADDED,␣
↪NULL);
    lv_obj_add_flag(chart, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);
    lv_obj_set_style_line_width(chart, 0, LV_PART_ITEMS);   /*Remove the␣
↪lines*/

    lv_chart_set_type(chart, LV_CHART_TYPE_SCATTER);

    lv_chart_set_axis_range(chart, LV_CHART_AXIS_PRIMARY_X, 0, 200);
    lv_chart_set_axis_range(chart, LV_CHART_AXIS_PRIMARY_Y, 0, 1000);

    lv_chart_set_point_count(chart, 50);

    lv_chart_series_t * ser = lv_chart_add_series(chart, lv_palette_
↪main(LV_PALETTE_RED), LV_CHART_AXIS_PRIMARY_Y);
    uint32_t i;
    for(i = 0; i < 50; i++) {
        lv_chart_set_next_value2(chart, ser, (int32_t)lv_rand(0, 200),␣
↪(int32_t)lv_rand(0, 1000));
    }

    lv_timer_create(add_data, 100, chart);
}
```

(continues on next page)

(continua dalla pagina precedente)

```
#endif
```

Grafico a linee circolari con spazio

```
#include "../../lv_examples.h"
#if LV_USE_CHART && LV_DRAW_SW_COMPLEX && LV_BUILD_EXAMPLES

static void add_data(lv_timer_t * t)
{
    lv_obj_t * chart = (lv_obj_t *)lv_timer_get_user_data(t);
    lv_chart_series_t * ser = lv_chart_get_series_next(chart, NULL);

    lv_chart_set_next_value(chart, ser, (int32_t)lv_rand(10, 90));

    uint32_t p = lv_chart_get_point_count(chart);
    uint32_t s = lv_chart_get_x_start_point(chart, ser);
    int32_t * a = lv_chart_get_series_y_array(chart, ser);

    a[(s + 1) % p] = LV_CHART_POINT_NONE;
    a[(s + 2) % p] = LV_CHART_POINT_NONE;
    a[(s + 2) % p] = LV_CHART_POINT_NONE;

    lv_chart_refresh(chart);
}

/**
 * Circular line chart with gap
 */
void lv_example_chart_8(void)
{
    /*Create a stacked_area_chart.obj*/
    lv_obj_t * chart = lv_chart_create(lv_screen_active());
    lv_chart_set_update_mode(chart, LV_CHART_UPDATE_MODE_CIRCULAR);
    lv_obj_set_style_size(chart, 0, 0, LV_PART_INDICATOR);
    lv_obj_set_size(chart, 280, 150);
    lv_obj_center(chart);

    lv_chart_set_point_count(chart, 80);
    lv_chart_series_t * ser = lv_chart_add_series(chart, lv_palette_
↪main(LV_PALETTE_RED), LV_CHART_AXIS_PRIMARY_Y);
    /*Prefill with data*/
    uint32_t i;
    for(i = 0; i < 80; i++) {
        lv_chart_set_next_value(chart, ser, (int32_t)lv_rand(10, 90));
    }

    lv_timer_create(add_data, 300, chart);

}

#endif
```

3.7.10 Checkbox

Checkbox Semplici

```
#include "../../lv_examples.h"
#if LV_USE_CHECKBOX && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        LV_UNUSED(obj);
        const char * txt = lv_checkbox_get_text(obj);
        const char * state = lv_obj_get_state(obj) & LV_STATE_CHECKED ?
↪"Checked" : "Unchecked";
        LV_UNUSED(txt);
        LV_UNUSED(state);
        LV_LOG_USER("%s: %s", txt, state);
    }
}

void lv_example_checkbox_1(void)
{
    lv_obj_set_flex_flow(lv_screen_active(), LV_FLEX_FLOW_COLUMN);
    lv_obj_set_flex_align(lv_screen_active(), LV_FLEX_ALIGN_CENTER, LV_
↪FLEX_ALIGN_START, LV_FLEX_ALIGN_CENTER);

    lv_obj_t * cb;
    cb = lv_checkbox_create(lv_screen_active());
    lv_checkbox_set_text(cb, "Apple");
    lv_obj_add_event_cb(cb, event_handler, LV_EVENT_ALL, NULL);

    cb = lv_checkbox_create(lv_screen_active());
    lv_checkbox_set_text(cb, "Banana");
    lv_obj_add_state(cb, LV_STATE_CHECKED);
    lv_obj_add_event_cb(cb, event_handler, LV_EVENT_ALL, NULL);

    cb = lv_checkbox_create(lv_screen_active());
    lv_checkbox_set_text(cb, "Lemon");
    lv_obj_add_state(cb, LV_STATE_DISABLED);
    lv_obj_add_event_cb(cb, event_handler, LV_EVENT_ALL, NULL);

    cb = lv_checkbox_create(lv_screen_active());
    lv_obj_add_state(cb, LV_STATE_CHECKED);
    lv_obj_add_state(cb, LV_STATE_DISABLED);
    lv_checkbox_set_text(cb, "Melon\nand a new line");
    lv_obj_add_event_cb(cb, event_handler, LV_EVENT_ALL, NULL);

    lv_obj_update_layout(cb);
}

#endif
```

Checkboxes come come radio button

```
#include "../../lv_examples.h"
#if LV_USE_CHECKBOX && LV_BUILD_EXAMPLES

static lv_style_t style_radio;
static lv_style_t style_radio_chk;
static int32_t active_index_1 = 0;
static int32_t active_index_2 = 0;

static void radio_event_handler(lv_event_t * e)
{
    int32_t * active_id = (int32_t *)lv_event_get_user_data(e);
    lv_obj_t * cont = (lv_obj_t *)lv_event_get_current_target(e);
    lv_obj_t * act_cb = lv_event_get_target_obj(e);
    lv_obj_t * old_cb = lv_obj_get_child(cont, *active_id);

    /*Do nothing if the container was clicked*/
    if(act_cb == cont) return;

    lv_obj_remove_state(old_cb, LV_STATE_CHECKED);   /*Uncheck the␣
↪previous radio button*/
    lv_obj_add_state(act_cb, LV_STATE_CHECKED);     /*Check the current␣
↪radio button*/

    *active_id = lv_obj_get_index(act_cb);

    LV_LOG_USER("Selected radio buttons: %d, %d", (int)active_index_1,␣
↪(int)active_index_2);
}

static void radiobutton_create(lv_obj_t * parent, const char * txt)
{
    lv_obj_t * obj = lv_checkbox_create(parent);
    lv_checkbox_set_text(obj, txt);
    lv_obj_add_flag(obj, LV_OBJ_FLAG_EVENT_BUBBLE);
    lv_obj_add_style(obj, &style_radio, LV_PART_INDICATOR);
    lv_obj_add_style(obj, &style_radio_chk, LV_PART_INDICATOR | LV_STATE_
↪CHECKED);
}

/**
 * Checkboxes as radio buttons
 */
void lv_example_checkbox_2(void)
{
    /* The idea is to enable `LV_OBJ_FLAG_EVENT_BUBBLE` on checkboxes and␣
↪process the
     * `LV_EVENT_CLICKED` on the container.
     * A variable is passed as event user data where the index of the␣
↪active
     * radiobutton is saved */

    lv_style_init(&style_radio);
    lv_style_set_radius(&style_radio, LV_RADIUS_CIRCLE);

    lv_style_init(&style_radio_chk);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_set_bg_image_src(&style_radio_chk, NULL);

    uint32_t i;
    char buf[32];

    lv_obj_t * cont1 = lv_obj_create(lv_screen_active());
    lv_obj_set_flex_flow(cont1, LV_FLEX_FLOW_COLUMN);
    lv_obj_set_size(cont1, lv_pct(40), lv_pct(80));
    lv_obj_add_event_cb(cont1, radio_event_handler, LV_EVENT_CLICKED, &
↪active_index_1);

    for(i = 0; i < 5; i++) {
        lv_snprintf(buf, sizeof(buf), "A %d", (int)i + 1);
        radiobutton_create(cont1, buf);

    }
    /*Make the first checkbox checked*/
    lv_obj_add_state(lv_obj_get_child(cont1, 0), LV_STATE_CHECKED);

    lv_obj_t * cont2 = lv_obj_create(lv_screen_active());
    lv_obj_set_flex_flow(cont2, LV_FLEX_FLOW_COLUMN);
    lv_obj_set_size(cont2, lv_pct(40), lv_pct(80));
    lv_obj_set_x(cont2, lv_pct(50));
    lv_obj_add_event_cb(cont2, radio_event_handler, LV_EVENT_CLICKED, &
↪active_index_2);

    for(i = 0; i < 3; i++) {
        lv_snprintf(buf, sizeof(buf), "B %d", (int)i + 1);
        radiobutton_create(cont2, buf);
    }

    /*Make the first checkbox checked*/
    lv_obj_add_state(lv_obj_get_child(cont2, 0), LV_STATE_CHECKED);
}

#endif
```

3.7.11 Menù a discesa

Menù a discesa semplice

```
#include "../../lv_examples.h"
#if LV_USE_DROPDOWN && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        char buf[32];
        lv_dropdown_get_selected_str(obj, buf, sizeof(buf));
        LV_LOG_USER("Option: %s", buf);
    }
}

void lv_example_dropdown_1(void)
```

(continues on next page)

(continua dalla pagina precedente)

```
{

    /*Create a normal drop down list*/
    lv_obj_t * dd = lv_dropdown_create(lv_screen_active());
    lv_dropdown_set_options(dd, "Apple\n"
                            "Banana\n"
                            "Orange\n"
                            "Cherry\n"
                            "Grape\n"
                            "Raspberry\n"
                            "Melon\n"
                            "Orange\n"
                            "Lemon\n"
                            "Nuts");

    lv_obj_align(dd, LV_ALIGN_TOP_MID, 0, 20);
    lv_obj_add_event_cb(dd, event_handler, LV_EVENT_ALL, NULL);
}


#endif
```

Menù a discesa nelle quattro direzioni

```
#include "../../lv_examples.h"
#if LV_USE_DROPDOWN && LV_BUILD_EXAMPLES

/**
 * Create a drop down, up, left and right menus
 */
void lv_example_dropdown_2(void)
{
    static const char * opts = "Apple\n"
                               "Banana\n"
                               "Orange\n"
                               "Melon";

    lv_obj_t * dd;
    dd = lv_dropdown_create(lv_screen_active());
    lv_dropdown_set_options_static(dd, opts);
    lv_obj_align(dd, LV_ALIGN_TOP_MID, 0, 10);

    dd = lv_dropdown_create(lv_screen_active());
    lv_dropdown_set_options_static(dd, opts);
    lv_dropdown_set_dir(dd, LV_DIR_BOTTOM);
    lv_dropdown_set_symbol(dd, LV_SYMBOL_UP);
    lv_obj_align(dd, LV_ALIGN_BOTTOM_MID, 0, -10);

    dd = lv_dropdown_create(lv_screen_active());
    lv_dropdown_set_options_static(dd, opts);
    lv_dropdown_set_dir(dd, LV_DIR_RIGHT);
    lv_dropdown_set_symbol(dd, LV_SYMBOL_RIGHT);
    lv_obj_align(dd, LV_ALIGN_LEFT_MID, 10, 0);

    dd = lv_dropdown_create(lv_screen_active());
    lv_dropdown_set_options_static(dd, opts);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_dropdown_set_dir(dd, LV_DIR_LEFT);
    lv_dropdown_set_symbol(dd, LV_SYMBOL_LEFT);
    lv_obj_align(dd, LV_ALIGN_RIGHT_MID, -10, 0);
}

#endif
```

Menù

```
#include "../../lv_examples.h"
#if LV_USE_DROPDOWN && LV_BUILD_EXAMPLES

static void event_cb(lv_event_t * e)
{
    lv_obj_t * dropdown = lv_event_get_target_obj(e);
    char buf[64];
    lv_dropdown_get_selected_str(dropdown, buf, sizeof(buf));
    LV_LOG_USER("'%s' is selected", buf);
}

/**
 * Create a menu from a drop-down list and show some drop-down list␣
 ↪features and styling
 */
void lv_example_dropdown_3(void)
{
    /*Create a drop down list*/
    lv_obj_t * dropdown = lv_dropdown_create(lv_screen_active());
    lv_obj_align(dropdown, LV_ALIGN_TOP_LEFT, 10, 10);
    lv_dropdown_set_options(dropdown, "New project\n"
                            "New file\n"
                            "Save\n"
                            "Save as ...\n"
                            "Open project\n"
                            "Recent projects\n"
                            "Preferences\n"
                            "Exit");

    /*Set a fixed text to display on the button of the drop-down list*/
    lv_dropdown_set_text(dropdown, "Menu");

    /*Use a custom image as down icon and flip it when the list is opened*/
    LV_IMAGE_DECLARE(img_caret_down);
    lv_dropdown_set_symbol(dropdown, &img_caret_down);
    lv_obj_set_style_transform_rotation(dropdown, 1800, LV_PART_INDICATOR␣
 ↪| LV_STATE_CHECKED);

    /*In a menu we don't need to show the last clicked item*/
    lv_dropdown_set_selected_highlight(dropdown, false);

    lv_obj_add_event_cb(dropdown, event_cb, LV_EVENT_VALUE_CHANGED, NULL);
}

#endif
```

3.7.12 Immagine

Immagine da variabile e simbolo

```
#include "../../lv_examples.h"
#if LV_USE_IMAGE && LV_BUILD_EXAMPLES

void lv_example_image_1(void)
{
    LV_IMAGE_DECLARE(img_cogwheel_argb);
    lv_obj_t * img1 = lv_image_create(lv_screen_active());
    lv_image_set_src(img1, &img_cogwheel_argb);
    lv_obj_align(img1, LV_ALIGN_CENTER, 0, 0);

    lv_obj_t * img2 = lv_image_create(lv_screen_active());
    lv_image_set_src(img2, LV_SYMBOL_OK "Accept");
    lv_obj_align_to(img2, img1, LV_ALIGN_OUT_BOTTOM_MID, 0, 20);
}

#endif
```

Ricolorazione dell'immagine

```
#include "../../lv_examples.h"
#if LV_USE_IMAGE && LV_USE_SLIDER && LV_BUILD_EXAMPLES

static lv_obj_t * create_slider(lv_color_t color);
static void slider_event_cb(lv_event_t * e);

static lv_obj_t * red_slider, * green_slider, * blue_slider, * intense_
↪slider;
static lv_obj_t * img1;

/**
 * Demonstrate runtime image re-coloring
 */
void lv_example_image_2(void)
{
    /*Create 4 sliders to adjust RGB color and re-color intensity*/
    red_slider = create_slider(lv_palette_main(LV_PALETTE_RED));
    green_slider = create_slider(lv_palette_main(LV_PALETTE_GREEN));
    blue_slider = create_slider(lv_palette_main(LV_PALETTE_BLUE));
    intense_slider = create_slider(lv_palette_main(LV_PALETTE_GREY));

    lv_slider_set_value(red_slider, LV_OPA_20, LV_ANIM_OFF);
    lv_slider_set_value(green_slider, LV_OPA_90, LV_ANIM_OFF);
    lv_slider_set_value(blue_slider, LV_OPA_60, LV_ANIM_OFF);
    lv_slider_set_value(intense_slider, LV_OPA_50, LV_ANIM_OFF);

    lv_obj_align(red_slider, LV_ALIGN_LEFT_MID, 25, 0);
    lv_obj_align_to(green_slider, red_slider, LV_ALIGN_OUT_RIGHT_MID, 25,␣
↪0);
    lv_obj_align_to(blue_slider, green_slider, LV_ALIGN_OUT_RIGHT_MID, 25,␣
↪0);
    lv_obj_align_to(intense_slider, blue_slider, LV_ALIGN_OUT_RIGHT_MID,␣
↪25, 0);
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Now create the actual image*/
    LV_IMAGE_DECLARE(img_cogwheel_argb);
    img1 = lv_image_create(lv_screen_active());
    lv_image_set_src(img1, &img_cogwheel_argb);
    lv_obj_align(img1, LV_ALIGN_RIGHT_MID, -20, 0);

    lv_obj_send_event(intense_slider, LV_EVENT_VALUE_CHANGED, NULL);
}

static void slider_event_cb(lv_event_t * e)
{
    LV_UNUSED(e);

    /*Recolor the image based on the sliders' values*/
    lv_color_t color  = lv_color_make((uint8_t)lv_slider_get_value(red_
↪slider),
                                      (uint8_t)lv_slider_get_value(green_
↪slider),
                                      (uint8_t)lv_slider_get_value(blue_
↪slider));
    lv_opa_t intense = (lv_opa_t)lv_slider_get_value(intense_slider);
    lv_obj_set_style_image_recolor_opa(img1, intense, 0);
    lv_obj_set_style_image_recolor(img1, color, 0);
}

static lv_obj_t * create_slider(lv_color_t color)
{
    lv_obj_t * slider = lv_slider_create(lv_screen_active());
    lv_slider_set_range(slider, 0, 255);
    lv_obj_set_size(slider, 10, 200);
    lv_obj_set_style_bg_color(slider, color, LV_PART_KNOB);
    lv_obj_set_style_bg_color(slider, lv_color_darken(color, LV_OPA_40),␣
↪LV_PART_INDICATOR);
    lv_obj_add_event_cb(slider, slider_event_cb, LV_EVENT_VALUE_CHANGED,␣
↪NULL);
    return slider;
}

#endif
```

Ruota e zoom

```
#include "../../lv_examples.h"
#if LV_USE_IMAGE && LV_BUILD_EXAMPLES

static void set_angle(void * img, int32_t v)
{
    lv_image_set_rotation((lv_obj_t *)img, v);
}

static void set_scale(void * img, int32_t v)
{
    lv_image_set_scale((lv_obj_t *)img, v);
}

/**
```

(continues on next page)

(continua dalla pagina precedente)

```
 * Show transformations (zoom and rotation) using a pivot point.
 */
void lv_example_image_3(void)
{
    LV_IMAGE_DECLARE(img_cogwheel_argb);

    /*Now create the actual image*/
    lv_obj_t * img = lv_image_create(lv_screen_active());
    lv_image_set_src(img, &img_cogwheel_argb);
    lv_obj_align(img, LV_ALIGN_CENTER, 50, 50);
    lv_image_set_pivot(img, 0, 0);    /*Rotate around the top left corner*/

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, img);
    lv_anim_set_exec_cb(&a, set_angle);
    lv_anim_set_values(&a, 0, 3600);
    lv_anim_set_duration(&a, 5000);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);
    lv_anim_start(&a);

    lv_anim_set_exec_cb(&a, set_scale);
    lv_anim_set_values(&a, 128, 256);
    lv_anim_set_reverse_duration(&a, 3000);
    lv_anim_start(&a);
}

#endif
```

Offset e stile dell'immagine

```
#include "../../lv_examples.h"
#if LV_USE_IMAGE && LV_BUILD_EXAMPLES

static void ofs_y_anim(void * img, int32_t v)
{
    lv_image_set_offset_y((lv_obj_t *)img, v);
}

/**
 * Image styling and offset
 */
void lv_example_image_4(void)
{
    LV_IMAGE_DECLARE(img_skew_strip);

    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_bg_color(&style, lv_palette_main(LV_PALETTE_YELLOW));
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_image_recolor_opa(&style, LV_OPA_COVER);
    lv_style_set_image_recolor(&style, lv_color_black());

    lv_obj_t * img = lv_image_create(lv_screen_active());
    lv_obj_add_style(img, &style, 0);
    lv_image_set_src(img, &img_skew_strip);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_size(img, 150, 100);
    lv_obj_center(img);

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, img);
    lv_anim_set_exec_cb(&a, ofs_y_anim);
    lv_anim_set_values(&a, 0, 100);
    lv_anim_set_duration(&a, 3000);
    lv_anim_set_reverse_duration(&a, 500);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);
    lv_anim_start(&a);

}

#endif
```

3.7.13 Pulsante Immagine

Pulsante immagine semplice

```
#include "../../lv_examples.h"
#if LV_USE_IMAGEBUTTON && LV_BUILD_EXAMPLES

void lv_example_imagebutton_1(void)
{
    LV_IMAGE_DECLARE(imagebutton_left);
    LV_IMAGE_DECLARE(imagebutton_right);
    LV_IMAGE_DECLARE(imagebutton_mid);

    /*Create a transition animation on width transformation and recolor.*/
    static lv_style_prop_t tr_prop[] = {LV_STYLE_TRANSFORM_WIDTH, LV_STYLE_
↪IMAGE_RECOLOR_OPA, 0};
    static lv_style_transition_dsc_t tr;
    lv_style_transition_dsc_init(&tr, tr_prop, lv_anim_path_linear, 200, 0,
↪ NULL);

    static lv_style_t style_def;
    lv_style_init(&style_def);
    lv_style_set_text_color(&style_def, lv_color_white());
    lv_style_set_transition(&style_def, &tr);

    /*Darken the button when pressed and make it wider*/
    static lv_style_t style_pr;
    lv_style_init(&style_pr);
    lv_style_set_image_recolor_opa(&style_pr, LV_OPA_30);
    lv_style_set_image_recolor(&style_pr, lv_color_black());
    lv_style_set_transform_width(&style_pr, 20);

    /*Create an image button*/
    lv_obj_t * imagebutton1 = lv_imagebutton_create(lv_screen_active());
    lv_imagebutton_set_src(imagebutton1, LV_IMAGEBUTTON_STATE_RELEASED, &
↪imagebutton_left, &imagebutton_mid,
                           &imagebutton_right);
    lv_obj_add_style(imagebutton1, &style_def, 0);
    lv_obj_add_style(imagebutton1, &style_pr, LV_STATE_PRESSED);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_width(imagebutton1, 100);
    lv_obj_align(imagebutton1, LV_ALIGN_CENTER, 0, 0);

    /*Create a label on the image button*/
    lv_obj_t * label = lv_label_create(imagebutton1);
    lv_label_set_text(label, "Button");
    lv_obj_align(label, LV_ALIGN_CENTER, 0, -4);
}

#endif
```

3.7.14 Tastiera

Tastiera con area di testo

```
#include "../../lv_examples.h"
#if LV_USE_KEYBOARD && LV_BUILD_EXAMPLES

static void ta_event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * ta = lv_event_get_target_obj(e);
    lv_obj_t * kb = (lv_obj_t *)lv_event_get_user_data(e);
    if(code == LV_EVENT_FOCUSED) {
        lv_keyboard_set_textarea(kb, ta);
        lv_obj_remove_flag(kb, LV_OBJ_FLAG_HIDDEN);
    }

    if(code == LV_EVENT_DEFOCUSED) {
        lv_keyboard_set_textarea(kb, NULL);
        lv_obj_add_flag(kb, LV_OBJ_FLAG_HIDDEN);
    }
}

void lv_example_keyboard_1(void)
{
    /*Create a keyboard to use it with an of the text areas*/
    lv_obj_t * kb = lv_keyboard_create(lv_screen_active());

    /*Create a text area. The keyboard will write here*/
    lv_obj_t * ta1;
    ta1 = lv_textarea_create(lv_screen_active());
    lv_obj_align(ta1, LV_ALIGN_TOP_LEFT, 10, 10);
    lv_obj_add_event_cb(ta1, ta_event_cb, LV_EVENT_ALL, kb);
    lv_textarea_set_placeholder_text(ta1, "Hello");
    lv_obj_set_size(ta1, 140, 80);

    lv_obj_t * ta2;
    ta2 = lv_textarea_create(lv_screen_active());
    lv_obj_align(ta2, LV_ALIGN_TOP_RIGHT, -10, 10);
    lv_obj_add_event_cb(ta2, ta_event_cb, LV_EVENT_ALL, kb);
    lv_obj_set_size(ta2, 140, 80);

    lv_keyboard_set_textarea(kb, ta1);
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*The keyboard will show Arabic characters if they are enabled */
#if LV_USE_ARABIC_PERSIAN_CHARS && LV_FONT_DEJAVU_16_PERSIAN_HEBREW
    lv_obj_set_style_text_font(kb, &lv_font_dejavu_16_persian_hebrew, 0);
    lv_obj_set_style_text_font(ta1, &lv_font_dejavu_16_persian_hebrew, 0);
    lv_obj_set_style_text_font(ta2, &lv_font_dejavu_16_persian_hebrew, 0);
#endif
}
#endif
```

Tastiera con mappa personalizzata

```
#include "../../lv_examples.h"
#if LV_USE_KEYBOARD && LV_BUILD_EXAMPLES

void lv_example_keyboard_2(void)
{
    /*Create an AZERTY keyboard map*/
    static const char * kb_map[] = {"A", "Z", "E", "R", "T", "Y", "U", "I",
 ↪ "O", "P", LV_SYMBOL_BACKSPACE, "\n",
                                    "Q", "S", "D", "F", "G", "J", "K", "L",
 ↪ "M",  LV_SYMBOL_NEW_LINE, "\n",
                                    "W", "X", "C", "V", "B", "N", ",", ".",
 ↪ ":", "!", "?", "\n",
                                    LV_SYMBOL_CLOSE, " ",  " ", " ", LV_
 ↪SYMBOL_OK, NULL
                                   };

    /*Set the relative width of the buttons and other controls*/
    static const lv_buttonmatrix_ctrl_t kb_ctrl[] = {LV_BUTTONMATRIX_CTRL_
 ↪WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_
 ↪BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_
 ↪CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4,
 ↪ LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_
 ↪BUTTONMATRIX_CTRL_WIDTH_6,
                                                        LV_BUTTONMATRIX_CTRL_
 ↪WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_
 ↪BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_
 ↪CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4,
 ↪ LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_6,
                                                        LV_BUTTONMATRIX_CTRL_
 ↪WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_
 ↪BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_
 ↪CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4,
 ↪ LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_
 ↪BUTTONMATRIX_CTRL_WIDTH_4,
                                                        LV_BUTTONMATRIX_CTRL_
 ↪WIDTH_2, (lv_buttonmatrix_ctrl_t)(LV_BUTTONMATRIX_CTRL_HIDDEN | LV_
 ↪BUTTONMATRIX_CTRL_WIDTH_2), LV_BUTTONMATRIX_CTRL_WIDTH_6, (lv_
 ↪buttonmatrix_ctrl_t)(LV_BUTTONMATRIX_CTRL_HIDDEN | LV_BUTTONMATRIX_CTRL_
 ↪WIDTH_2), LV_BUTTONMATRIX_CTRL_WIDTH_2
                                                       };

    /*Create a keyboard and add the new map as USER_1 mode*/
    lv_obj_t * kb = lv_keyboard_create(lv_screen_active());

    lv_keyboard_set_map(kb, LV_KEYBOARD_MODE_USER_1, kb_map, kb_ctrl);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_keyboard_set_mode(kb, LV_KEYBOARD_MODE_USER_1);

    /*Create a text area. The keyboard will write here*/
    lv_obj_t * ta;
    ta = lv_textarea_create(lv_screen_active());
    lv_obj_align(ta, LV_ALIGN_TOP_MID, 0, 10);
    lv_obj_set_size(ta, lv_pct(90), 80);
    lv_obj_add_state(ta, LV_STATE_FOCUSED);

    lv_keyboard_set_textarea(kb, ta);
}
#endif
```

Tastiera con disegno

```
#include "../../lv_examples.h"
#if LV_USE_BUTTONMATRIX && LV_BUILD_EXAMPLES

static void event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);
    bool pressed = false;
    if(lv_keyboard_get_selected_button(obj) == base_dsc->id1 && lv_obj_has_
↪state(obj, LV_STATE_PRESSED)) {
        pressed = true;
    }

    /*When the keyboard draws the buttons...*/
    if(base_dsc->part == LV_PART_ITEMS) {
        /*Get a color based on the button's index*/
        lv_palette_t palette = (lv_palette_t)(base_dsc->id1 % LV_PALETTE_
↪LAST);
        lv_draw_fill_dsc_t * fill_draw_dsc = lv_draw_task_get_fill_
↪dsc(draw_task);
        if(fill_draw_dsc) {
            fill_draw_dsc->color = pressed ? lv_palette_darken(palette, 3)␣
↪: lv_palette_main(palette);
        }
        lv_draw_label_dsc_t * label_draw_dsc = lv_draw_task_get_label_
↪dsc(draw_task);
        if(label_draw_dsc) {
            /*For the OK symbol, draw a star instead*/
            if(lv_strcmp(label_draw_dsc->text, LV_SYMBOL_OK) == 0) {
                label_draw_dsc->opa = 0;    /*Hide the label*/

                LV_IMAGE_DECLARE(img_star);
                lv_image_header_t header;
                lv_result_t res = lv_image_decoder_get_info(&img_star, &
↪header);
                if(res != LV_RESULT_OK) return;

                lv_area_t a;
                lv_area_set(&a, 0, 0, header.w - 1, header.h - 1);
```

(continues on next page)

(continua dalla pagina precedente)

```
                lv_area_t draw_task_area;
                lv_draw_task_get_area(draw_task, &draw_task_area);
                lv_area_align(&draw_task_area, &a, LV_ALIGN_CENTER, 0, 0);

                lv_draw_image_dsc_t img_draw_dsc;
                lv_draw_image_dsc_init(&img_draw_dsc);
                img_draw_dsc.src = &img_star;
                lv_draw_image(base_dsc->layer, &img_draw_dsc, &a);
            }
            else {
                /*For the other labels just pick an lighter color*/
                label_draw_dsc->color = lv_palette_lighten(palette, 4);
            }
        }

    }
}

/**
 * Add custom drawer to the keyboard to customize the buttons one by one
 */
void lv_example_keyboard_3(void)
{
    lv_obj_t * kb = lv_keyboard_create(lv_screen_active());
    lv_obj_add_event_cb(kb, event_cb, LV_EVENT_DRAW_TASK_ADDED, NULL);
    lv_obj_add_flag(kb, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);
    lv_obj_center(kb);
}

#endif
```

3.7.15 Etichetta

A capo automatico, ricolorazione e scorrimento

```
#include "../../lv_examples.h"
#if LV_USE_LABEL && LV_BUILD_EXAMPLES

/**
 * Show line wrap, re-color, line align and text scrolling.
 */
void lv_example_label_1(void)
{
    lv_obj_t * label1 = lv_label_create(lv_screen_active());
    lv_label_set_long_mode(label1, LV_LABEL_LONG_MODE_WRAP);     /*Break␣
↪the long lines*/
    lv_label_set_recolor(label1, true);                      /*Enable re-
↪coloring by commands in the text*/
    lv_label_set_text(label1, "#0000ff Re-color# #ff00ff words# #ff0000 of␣
↪a# label, align the lines to the center "
                      "and wrap long text automatically.");
    lv_obj_set_width(label1, 150);  /*Set smaller width to make the lines␣
↪wrap*/
    lv_obj_set_style_text_align(label1, LV_TEXT_ALIGN_CENTER, 0);
    lv_obj_align(label1, LV_ALIGN_CENTER, 0, -40);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_t * label2 = lv_label_create(lv_screen_active());
    lv_label_set_long_mode(label2, LV_LABEL_LONG_MODE_SCROLL_CIRCULAR);
↪ /*Circular scroll*/
    lv_obj_set_width(label2, 150);
    lv_label_set_text(label2, "It is a circularly scrolling text. ");
    lv_obj_align(label2, LV_ALIGN_CENTER, 0, 40);
}

#endif
```

Ombreggiatura del testo

```
#include "../../lv_examples.h"
#if LV_USE_LABEL && LV_BUILD_EXAMPLES

/**
 * Create a fake text shadow
 */
void lv_example_label_2(void)
{
    /*Create a style for the shadow*/
    static lv_style_t style_shadow;
    lv_style_init(&style_shadow);
    lv_style_set_text_opa(&style_shadow, LV_OPA_30);
    lv_style_set_text_color(&style_shadow, lv_color_black());

    /*Create a label for the shadow first (it's in the background)*/
    lv_obj_t * shadow_label = lv_label_create(lv_screen_active());
    lv_obj_add_style(shadow_label, &style_shadow, 0);

    /*Create the main label*/
    lv_obj_t * main_label = lv_label_create(lv_screen_active());
    lv_label_set_text(main_label, "A simple method to create\n"
                      "shadows on a text.\n"
                      "It even works with\n\n"
                      "newlines     and spaces.");

    /*Set the same text for the shadow label*/
    lv_label_set_text(shadow_label, lv_label_get_text(main_label));

    /*Position the main label*/
    lv_obj_align(main_label, LV_ALIGN_CENTER, 0, 0);

    /*Shift the second label down and to the right by 2 pixel*/
    lv_obj_align_to(shadow_label, main_label, LV_ALIGN_TOP_LEFT, 2, 2);
}

#endif
```

Visualizzazione di testi LTR, RTL e in cinese

```
#include "../../lv_examples.h"
#if LV_USE_LABEL && LV_BUILD_EXAMPLES && LV_FONT_DEJAVU_16_PERSIAN_HEBREW &
↪& LV_FONT_SOURCE_HAN_SANS_SC_16_CJK && LV_USE_BIDI
```

(continues on next page)

(continua dalla pagina precedente)

```
/**
 * Show mixed LTR, RTL and Chinese label
 */
void lv_example_label_3(void)
{
    lv_obj_t * ltr_label = lv_label_create(lv_screen_active());
    lv_label_set_text(ltr_label, "In modern terminology, a microcontroller␣
↪is similar to a system on a chip (SoC).");
    lv_obj_set_style_text_font(ltr_label, &lv_font_montserrat_16, 0);
    lv_obj_set_width(ltr_label, 310);
    lv_obj_align(ltr_label, LV_ALIGN_TOP_LEFT, 5, 5);

    lv_obj_t * rtl_label = lv_label_create(lv_screen_active());
    lv_label_set_text(rtl_label,
                      "□□□□, □□ □□□□ □□□□ □□□□□ □□□□□ □□□□□□ (□□□□□□□: CPU␣
↪- Central Processing Unit).");
    lv_obj_set_style_base_dir(rtl_label, LV_BASE_DIR_RTL, 0);
    lv_obj_set_style_text_font(rtl_label, &lv_font_dejavu_16_persian_
↪hebrew, 0);
    lv_obj_set_width(rtl_label, 310);
    lv_obj_align(rtl_label, LV_ALIGN_LEFT_MID, 5, 0);

    lv_obj_t * cz_label = lv_label_create(lv_screen_active());
    lv_label_set_text(cz_label,
                      "□□□□□□Embedded System□□\
↪n□□□□□□□
```

(continua dalla pagina precedente)

```
    lv_draw_label_dsc_t label_dsc;
    lv_draw_label_dsc_init(&label_dsc);
    label_dsc.color = lv_color_white();
    label_dsc.align = LV_TEXT_ALIGN_CENTER;
    label_dsc.text = txt;
    label_dsc.font = &lv_font_montserrat_24;
    lv_area_t a = {0, 0, w - 1, h - 1};
    lv_draw_label(&layer, &label_dsc, &a);

    lv_canvas_finish_layer(canvas, &layer);

    lv_obj_delete(canvas);
}

/**
 * Draw label with gradient color
 */
void lv_example_label_4(void)
{
    /* Create the mask of a text by drawing it to a canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(mask, MASK_WIDTH, MASK_HEIGHT, LV_COLOR_
↪FORMAT_L8);
    LV_DRAW_BUF_INIT_STATIC(mask);

    generate_mask(&mask, MASK_WIDTH, MASK_HEIGHT, "Text with gradient");

    /* Create an object from where the text will be masked out.
     * Now it's a rectangle with a gradient but it could be an image too*/
    lv_obj_t * grad = lv_obj_create(lv_screen_active());
    lv_obj_set_size(grad, MASK_WIDTH, MASK_HEIGHT);
    lv_obj_center(grad);
    lv_obj_set_style_bg_color(grad, lv_color_hex(0xff0000), 0);
    lv_obj_set_style_bg_grad_color(grad, lv_color_hex(0x0000ff), 0);
    lv_obj_set_style_bg_grad_dir(grad, LV_GRAD_DIR_HOR, 0);
    lv_obj_set_style_bitmap_mask_src(grad, &mask, 0);
}

#endif
```

Personalizzazione dell'animazione a scorrimento circolare

```
#include "../../lv_examples.h"
#if LV_USE_LABEL && LV_BUILD_EXAMPLES

/**
 * Show customizing the circular scrolling animation of a label with `LV_
↪LABEL_LONG_MODE_SCROLL_CIRCULAR`
 * long mode.
 */
void lv_example_label_5(void)
{
    static lv_anim_t animation_template;
    static lv_style_t label_style;

    lv_anim_init(&animation_template);
    lv_anim_set_delay(&animation_template, 1000);           /*Wait 1␣
```

(continues on next page)

(continua dalla pagina precedente)

```
↪second to start the first scroll*/
    lv_anim_set_repeat_delay(&animation_template,
                             3000);    /*Repeat the scroll 3 seconds after␣
↪the label scrolls back to the initial position*/
    lv_anim_set_repeat_count(&animation_template, LV_ANIM_REPEAT_INFINITE);

    /*Initialize the label style with the animation template*/
    lv_style_init(&label_style);
    lv_style_set_anim(&label_style, &animation_template);

    lv_obj_t * label1 = lv_label_create(lv_screen_active());
    lv_label_set_long_mode(label1, LV_LABEL_LONG_MODE_SCROLL_CIRCULAR);    ␣
↪   /*Circular scroll*/
    lv_obj_set_width(label1, 150);
    lv_label_set_text(label1, "It is a circularly scrolling text. ");
    lv_obj_align(label1, LV_ALIGN_CENTER, 0, 40);
    lv_obj_add_style(label1, &label_style, LV_STATE_DEFAULT);           /
↪*Add the style to the label*/
}

#endif
```

Carattere a spaziatura fissa

```
#include "../../lv_examples.h"
#if LV_USE_LABEL && LV_BUILD_EXAMPLES && LV_FONT_MONTSERRAT_20

static bool fix_w_get_glyph_dsc(const lv_font_t * font, lv_font_glyph_dsc_
↪t * dsc, uint32_t letter,
                                uint32_t letter_next)
{
    bool ret = lv_font_get_glyph_dsc_fmt_txt(font, dsc, letter, letter_
↪next);
    if(!ret) return false;

    /* Set a fixed width */
    dsc->adv_w = 20;
    dsc->ofs_x = (dsc->adv_w - dsc->box_w) / 2;
    return true;
}

void lv_example_label_6(void)
{
    /* Clone the original font and override its behavior */
    static lv_font_t mono_font;
    mono_font = lv_font_montserrat_20;
    mono_font.get_glyph_dsc = fix_w_get_glyph_dsc;

    /* Create a label with normal font */
    lv_obj_t * label1 = lv_label_create(lv_screen_active());
    lv_obj_set_style_text_font(label1, &lv_font_montserrat_20, 0);
    lv_label_set_text(label1, "0123.Wabc");

    /* Create a label with fixed-width glyph descriptor override */
    lv_obj_t * label2 = lv_label_create(lv_screen_active());
    lv_obj_set_y(label2, 30);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_style_text_font(label2, &mono_font, 0);
    lv_label_set_text(label2, "0123.Wabc");
}

#endif
```

3.7.16 LED

LED con stile personalizzato

```
#include "../../lv_examples.h"
#if LV_USE_LED && LV_BUILD_EXAMPLES

/**
 * Create LED's with different brightness and color
 */
void lv_example_led_1(void)
{
    /*Create a LED and switch it OFF*/
    lv_obj_t * led1  = lv_led_create(lv_screen_active());
    lv_obj_align(led1, LV_ALIGN_CENTER, -80, 0);
    lv_led_off(led1);

    /*Copy the previous LED and set a brightness*/
    lv_obj_t * led2  = lv_led_create(lv_screen_active());
    lv_obj_align(led2, LV_ALIGN_CENTER, 0, 0);
    lv_led_set_brightness(led2, 150);
    lv_led_set_color(led2, lv_palette_main(LV_PALETTE_RED));

    /*Copy the previous LED and switch it ON*/
    lv_obj_t * led3  = lv_led_create(lv_screen_active());
    lv_obj_align(led3, LV_ALIGN_CENTER, 80, 0);
    lv_led_on(led3);
}

#endif
```

3.7.17 Linea

Linea semplice

```
#include "../../lv_examples.h"
#if LV_USE_LINE && LV_BUILD_EXAMPLES

void lv_example_line_1(void)
{
    /*Create an array for the points of the line*/
    static lv_point_precise_t line_points[] = { {5, 5}, {70, 70}, {120, 10}
↪, {180, 60}, {240, 10} };

    /*Create style*/
    static lv_style_t style_line;
    lv_style_init(&style_line);
    lv_style_set_line_width(&style_line, 8);
    lv_style_set_line_color(&style_line, lv_palette_main(LV_PALETTE_BLUE));
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_set_line_rounded(&style_line, true);

    /*Create a line and apply the new style*/
    lv_obj_t * line1;
    line1 = lv_line_create(lv_screen_active());
    lv_line_set_points(line1, line_points, 5);     /*Set the points*/
    lv_obj_add_style(line1, &style_line, 0);
    lv_obj_center(line1);
}

#endif
```

3.7.18 Lista

Lista semplice

```
#include "../../lv_examples.h"
#if LV_USE_LIST && LV_BUILD_EXAMPLES
static lv_obj_t * list1;

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_CLICKED) {
        LV_UNUSED(obj);
        LV_LOG_USER("Clicked: %s", lv_list_get_button_text(list1, obj));
    }
}
void lv_example_list_1(void)
{
    /*Create a list*/
    list1 = lv_list_create(lv_screen_active());
    lv_obj_set_size(list1, 180, 220);
    lv_obj_center(list1);

    /*Add buttons to the list*/
    lv_obj_t * btn;
    lv_list_add_text(list1, "File");
    btn = lv_list_add_button(list1, LV_SYMBOL_FILE, "New");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);
    btn = lv_list_add_button(list1, LV_SYMBOL_DIRECTORY, "Open");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);
    btn = lv_list_add_button(list1, LV_SYMBOL_SAVE, "Save");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);
    btn = lv_list_add_button(list1, LV_SYMBOL_CLOSE, "Delete");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);
    btn = lv_list_add_button(list1, LV_SYMBOL_EDIT, "Edit");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);

    lv_list_add_text(list1, "Connectivity");
    btn = lv_list_add_button(list1, LV_SYMBOL_BLUETOOTH, "Bluetooth");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);
    btn = lv_list_add_button(list1, LV_SYMBOL_GPS, "Navigation");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);
    btn = lv_list_add_button(list1, LV_SYMBOL_USB, "USB");
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);
    btn = lv_list_add_button(list1, LV_SYMBOL_BATTERY_FULL, "Battery");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);

    lv_list_add_text(list1, "Exit");
    btn = lv_list_add_button(list1, LV_SYMBOL_OK, "Apply");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);
    btn = lv_list_add_button(list1, LV_SYMBOL_CLOSE, "Close");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);
}

#endif
```

Ordinamento di un elenco utilizzando i pulsanti Su e Giù

```
#include "../../lv_examples.h"
#if LV_USE_LIST && LV_BUILD_EXAMPLES

static lv_obj_t * list1;
static lv_obj_t * list2;

static lv_obj_t * currentButton = NULL;

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_CLICKED) {
        LV_LOG_USER("Clicked: %s", lv_list_get_button_text(list1, obj));

        if(currentButton == obj) {
            currentButton = NULL;
        }
        else {
            currentButton = obj;
        }
        lv_obj_t * parent = lv_obj_get_parent(obj);
        int32_t i;
        for(i = 0; i < (int32_t)lv_obj_get_child_count(parent); i++) {
            lv_obj_t * child = lv_obj_get_child(parent, i);
            if(child == currentButton) {
                lv_obj_add_state(child, LV_STATE_CHECKED);
            }
            else {
                lv_obj_remove_state(child, LV_STATE_CHECKED);
            }
        }
    }
}

static void event_handler_top(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    if(code == LV_EVENT_CLICKED) {
        if(currentButton == NULL) return;
        lv_obj_move_background(currentButton);
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_obj_scroll_to_view(currentButton, LV_ANIM_ON);
    }
}

static void event_handler_up(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    if((code == LV_EVENT_CLICKED) || (code == LV_EVENT_LONG_PRESSED_
↪REPEAT)) {
        if(currentButton == NULL) return;
        int32_t index = lv_obj_get_index(currentButton);
        if(index <= 0) return;
        lv_obj_move_to_index(currentButton, index - 1);
        lv_obj_scroll_to_view(currentButton, LV_ANIM_ON);
    }
}

static void event_handler_center(lv_event_t * e)
{
    const lv_event_code_t code = lv_event_get_code(e);
    if((code == LV_EVENT_CLICKED) || (code == LV_EVENT_LONG_PRESSED_
↪REPEAT)) {
        if(currentButton == NULL) return;

        lv_obj_t * parent = lv_obj_get_parent(currentButton);
        const int32_t pos = (int32_t)lv_obj_get_child_count(parent) / 2;

        lv_obj_move_to_index(currentButton, pos);

        lv_obj_scroll_to_view(currentButton, LV_ANIM_ON);
    }
}

static void event_handler_dn(lv_event_t * e)
{
    const lv_event_code_t code = lv_event_get_code(e);
    if((code == LV_EVENT_CLICKED) || (code == LV_EVENT_LONG_PRESSED_
↪REPEAT)) {
        if(currentButton == NULL) return;
        const int32_t index = lv_obj_get_index(currentButton);

        lv_obj_move_to_index(currentButton, index + 1);
        lv_obj_scroll_to_view(currentButton, LV_ANIM_ON);
    }
}

static void event_handler_bottom(lv_event_t * e)
{
    const lv_event_code_t code = lv_event_get_code(e);
    if(code == LV_EVENT_CLICKED) {
        if(currentButton == NULL) return;
        lv_obj_move_foreground(currentButton);
        lv_obj_scroll_to_view(currentButton, LV_ANIM_ON);
    }
}
```

(continues on next page)

(continua dalla pagina precedente)

```
static void event_handler_swap(lv_event_t * e)
{
    const lv_event_code_t code = lv_event_get_code(e);
    if((code == LV_EVENT_CLICKED) || (code == LV_EVENT_LONG_PRESSED_
↪REPEAT)) {
        uint32_t cnt = lv_obj_get_child_count(list1);
        for(int i = 0; i < 100; i++) {
            lv_obj_t * obj = lv_obj_get_child(list1, (int32_t)lv_rand(0,␣
↪cnt - 1));
            lv_obj_move_to_index(obj, (int32_t)lv_rand(0, cnt - 1));
            if(currentButton != NULL) {
                lv_obj_scroll_to_view(currentButton, LV_ANIM_ON);
            }
        }
    }
}

void lv_example_list_2(void)
{
    /*Create a list*/
    list1 = lv_list_create(lv_screen_active());
    lv_obj_set_size(list1, lv_pct(60), lv_pct(100));
    lv_obj_set_style_pad_row(list1, 5, 0);

    /*Add buttons to the list*/
    lv_obj_t * btn;
    int i;
    for(i = 0; i < 15; i++) {
        btn = lv_button_create(list1);
        lv_obj_set_width(btn, lv_pct(50));
        lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);

        lv_obj_t * lab = lv_label_create(btn);
        lv_label_set_text_fmt(lab, "Item %d", i);
    }

    /*Select the first button by default*/
    currentButton = lv_obj_get_child(list1, 0);
    lv_obj_add_state(currentButton, LV_STATE_CHECKED);

    /*Create a second list with up and down buttons*/
    list2 = lv_list_create(lv_screen_active());
    lv_obj_set_size(list2, lv_pct(40), lv_pct(100));
    lv_obj_align(list2, LV_ALIGN_TOP_RIGHT, 0, 0);
    lv_obj_set_flex_flow(list2, LV_FLEX_FLOW_COLUMN);

    btn = lv_list_add_button(list2, NULL, "Top");
    lv_obj_add_event_cb(btn, event_handler_top, LV_EVENT_ALL, NULL);
    lv_group_remove_obj(btn);

    btn = lv_list_add_button(list2, LV_SYMBOL_UP, "Up");
    lv_obj_add_event_cb(btn, event_handler_up, LV_EVENT_ALL, NULL);
    lv_group_remove_obj(btn);

    btn = lv_list_add_button(list2, LV_SYMBOL_LEFT, "Center");
    lv_obj_add_event_cb(btn, event_handler_center, LV_EVENT_ALL, NULL);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_group_remove_obj(btn);

    btn = lv_list_add_button(list2, LV_SYMBOL_DOWN, "Down");
    lv_obj_add_event_cb(btn, event_handler_dn, LV_EVENT_ALL, NULL);
    lv_group_remove_obj(btn);

    btn = lv_list_add_button(list2, NULL, "Bottom");
    lv_obj_add_event_cb(btn, event_handler_bottom, LV_EVENT_ALL, NULL);
    lv_group_remove_obj(btn);

    btn = lv_list_add_button(list2, LV_SYMBOL_SHUFFLE, "Shuffle");
    lv_obj_add_event_cb(btn, event_handler_swap, LV_EVENT_ALL, NULL);
    lv_group_remove_obj(btn);
}

#endif
```

3.7.19 Lottie

Caricamento di un'animazione Lottie da un array

```
#include "../../lv_examples.h"
#if LV_BUILD_EXAMPLES
#if LV_USE_LOTTIE

/**
 * Load an lottie animation from data
 */
void lv_example_lottie_1(void)
{
    extern const uint8_t lv_example_lottie_approve[];
    extern const size_t lv_example_lottie_approve_size;

    lv_obj_t * lottie = lv_lottie_create(lv_screen_active());
    lv_lottie_set_src_data(lottie, lv_example_lottie_approve, lv_example_
↪lottie_approve_size);

#if LV_DRAW_BUF_ALIGN == 4 && LV_DRAW_BUF_STRIDE_ALIGN == 1
    /*If there are no special requirements, just declare a buffer
      x4 because the Lottie is rendered in ARGB8888_PREMULTIPLIED format*/
    static uint8_t buf[64 * 64 * 4];
    lv_lottie_set_buffer(lottie, 64, 64, buf);
#else
    /*For GPUs and special alignment/strid setting use a draw_buf instead*/
    LV_DRAW_BUF_DEFINE(draw_buf, 64, 64, LV_COLOR_FORMAT_ARGB8888_
↪PREMULTIPLIED);
    lv_lottie_set_draw_buf(lottie, &draw_buf);
#endif

    lv_obj_center(lottie);
}

#else

void lv_example_lottie_1(void)
{
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*fallback for online examples*/

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Lottie cannot be previewed online");
    lv_obj_center(label);
}

#endif /*LV_USE_LOTTIE*/

#endif /*LV_BUILD_EXAMPLES*/
```

Caricamento di un'animazione Lottie da un file

```
#include "../../lv_examples.h"
#if LV_BUILD_EXAMPLES
#if LV_USE_LOTTIE

/**
 * Load an lottie animation from file
 */
void lv_example_lottie_2(void)
{

    lv_obj_t * lottie = lv_lottie_create(lv_screen_active());
    lv_lottie_set_src_file(lottie, "lvgl/examples/widgets/lottie/lv_
↪example_lottie_approve.json");

#if LV_DRAW_BUF_ALIGN == 4 && LV_DRAW_BUF_STRIDE_ALIGN == 1
    /*If there are no special requirements, just declare a buffer
      x4 because the Lottie is rendered in ARGB8888_PREMULTIPLIED format*/
    static uint8_t buf[64 * 64 * 4];
    lv_lottie_set_buffer(lottie, 64, 64, buf);
#else
    /*For GPUs and special alignment/strid setting use a draw_buf instead*/
    LV_DRAW_BUF_DEFINE(draw_buf, 64, 64, LV_COLOR_FORMAT_ARGB8888_
↪PREMULTIPLIED);
    lv_lottie_set_draw_buf(lottie, &draw_buf);
#endif

    lv_obj_center(lottie);
}

#else

void lv_example_lottie_2(void)
{
    /*fallback for online examples*/

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Lottie cannot be previewed online");
    lv_obj_center(label);
}

#endif /*LV_USE_LOTTIE*/

#endif /*LV_BUILD_EXAMPLES*/
```

3.7.20 Menù

Menù semplice

```
#include "../../lv_examples.h"
#if LV_USE_MENU && LV_BUILD_EXAMPLES

void lv_example_menu_1(void)
{
    /*Create a menu object*/
    lv_obj_t * menu = lv_menu_create(lv_screen_active());
    lv_obj_set_size(menu, lv_display_get_horizontal_resolution(NULL), lv_
↪display_get_vertical_resolution(NULL));
    lv_obj_center(menu);

    lv_obj_t * cont;
    lv_obj_t * label;

    /*Create a sub page*/
    lv_obj_t * sub_page = lv_menu_page_create(menu, NULL);

    cont = lv_menu_cont_create(sub_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Hello, I am hiding here");

    /*Create a main page*/
    lv_obj_t * main_page = lv_menu_page_create(menu, NULL);

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 1");

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 2");

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 3 (Click me!)");
    lv_menu_set_load_page_event(menu, cont, sub_page);

    lv_menu_set_page(menu, main_page);
}

#endif
```

Menù semplice con pulsante principale

```
#include "../../lv_examples.h"
#if LV_USE_MENU && LV_USE_MSGBOX && LV_BUILD_EXAMPLES

static void back_event_handler(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    lv_obj_t * menu = (lv_obj_t *)lv_event_get_user_data(e);

    if(lv_menu_back_button_is_root(menu, obj)) {
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_obj_t * mbox1 = lv_msgbox_create(NULL);
        lv_msgbox_add_title(mbox1, "Hello");
        lv_msgbox_add_text(mbox1, "Root back btn click.");
        lv_msgbox_add_close_button(mbox1);
    }
}

void lv_example_menu_2(void)
{
    lv_obj_t * menu = lv_menu_create(lv_screen_active());
    lv_menu_set_mode_root_back_button(menu, LV_MENU_ROOT_BACK_BUTTON_
↪ENABLED);
    lv_obj_add_event_cb(menu, back_event_handler, LV_EVENT_CLICKED, menu);
    lv_obj_set_size(menu, lv_display_get_horizontal_resolution(NULL), lv_
↪display_get_vertical_resolution(NULL));
    lv_obj_center(menu);

    lv_obj_t * cont;
    lv_obj_t * label;

    /*Create a sub page*/
    lv_obj_t * sub_page = lv_menu_page_create(menu, NULL);

    cont = lv_menu_cont_create(sub_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Hello, I am hiding here");

    /*Create a main page*/
    lv_obj_t * main_page = lv_menu_page_create(menu, NULL);

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 1");

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 2");

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 3 (Click me!)");
    lv_menu_set_load_page_event(menu, cont, sub_page);

    lv_menu_set_page(menu, main_page);
}

#endif
```

Menù semplice con intestazione personalizzata

```
#include "../../lv_examples.h"
#if LV_USE_MENU && LV_BUILD_EXAMPLES

void lv_example_menu_3(void)
{
    /*Create a menu object*/
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_t * menu = lv_menu_create(lv_screen_active());
    lv_obj_set_size(menu, lv_display_get_horizontal_resolution(NULL), lv_
↪display_get_vertical_resolution(NULL));
    lv_obj_center(menu);

    /*Modify the header*/
    lv_obj_t * back_btn = lv_menu_get_main_header_back_button(menu);
    lv_obj_t * back_button_label = lv_label_create(back_btn);
    lv_label_set_text(back_button_label, "Back");

    lv_obj_t * cont;
    lv_obj_t * label;

    /*Create sub pages*/
    lv_obj_t * sub_1_page = lv_menu_page_create(menu, "Page 1");

    cont = lv_menu_cont_create(sub_1_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Hello, I am hiding here");

    lv_obj_t * sub_2_page = lv_menu_page_create(menu, "Page 2");

    cont = lv_menu_cont_create(sub_2_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Hello, I am hiding here");

    lv_obj_t * sub_3_page = lv_menu_page_create(menu, "Page 3");

    cont = lv_menu_cont_create(sub_3_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Hello, I am hiding here");

    /*Create a main page*/
    lv_obj_t * main_page = lv_menu_page_create(menu, NULL);

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 1 (Click me!)");
    lv_menu_set_load_page_event(menu, cont, sub_1_page);

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 2 (Click me!)");
    lv_menu_set_load_page_event(menu, cont, sub_2_page);

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 3 (Click me!)");
    lv_menu_set_load_page_event(menu, cont, sub_3_page);

    lv_menu_set_page(menu, main_page);
}

#endif
```

Menù semplice con pulsante mobile per aggiungere una nuova pagina di menu

```
#include "../../lv_examples.h"
#if LV_USE_MENU && LV_BUILD_EXAMPLES

static uint32_t btn_cnt = 1;
static lv_obj_t * main_page;
static lv_obj_t * menu;

static void float_button_event_cb(lv_event_t * e)
{
    LV_UNUSED(e);

    btn_cnt++;

    lv_obj_t * cont;
    lv_obj_t * label;

    lv_obj_t * sub_page = lv_menu_page_create(menu, NULL);

    cont = lv_menu_cont_create(sub_page);
    label = lv_label_create(cont);
    lv_label_set_text_fmt(label, "Hello, I am hiding inside %" LV_PRIu32"",
 ↪ btn_cnt);

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text_fmt(label, "Item %" LV_PRIu32"", btn_cnt);
    lv_menu_set_load_page_event(menu, cont, sub_page);

    lv_obj_scroll_to_view_recursive(cont, LV_ANIM_ON);
}

void lv_example_menu_4(void)
{
    /*Create a menu object*/
    menu = lv_menu_create(lv_screen_active());
    lv_obj_set_size(menu, lv_display_get_horizontal_resolution(NULL), lv_
 ↪display_get_vertical_resolution(NULL));
    lv_obj_center(menu);

    lv_obj_t * cont;
    lv_obj_t * label;

    /*Create a sub page*/
    lv_obj_t * sub_page = lv_menu_page_create(menu, NULL);

    cont = lv_menu_cont_create(sub_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Hello, I am hiding inside the first item");

    /*Create a main page*/
    main_page = lv_menu_page_create(menu, NULL);

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 1");
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_menu_set_load_page_event(menu, cont, sub_page);

    lv_menu_set_page(menu, main_page);

    /*Create floating btn*/
    lv_obj_t * float_btn = lv_button_create(lv_screen_active());
    lv_obj_set_size(float_btn, 50, 50);
    lv_obj_add_flag(float_btn, LV_OBJ_FLAG_FLOATING);
    lv_obj_align(float_btn, LV_ALIGN_BOTTOM_RIGHT, -10, -10);
    lv_obj_add_event_cb(float_btn, float_button_event_cb, LV_EVENT_CLICKED,
↪ menu);
    lv_obj_set_style_radius(float_btn, LV_RADIUS_CIRCLE, 0);
    lv_obj_set_style_bg_image_src(float_btn, LV_SYMBOL_PLUS, 0);
    lv_obj_set_style_text_font(float_btn, lv_theme_get_font_large(float_
↪btn), 0);
}

#endif
```

Menù complesso

```
#include "../../lv_examples.h"
#if LV_USE_MENU && LV_USE_MSGBOX && LV_BUILD_EXAMPLES

typedef enum {
    LV_MENU_ITEM_BUILDER_VARIANT_1,
    LV_MENU_ITEM_BUILDER_VARIANT_2
} lv_menu_builder_variant_t;

static void back_event_handler(lv_event_t * e);
static void switch_handler(lv_event_t * e);
static lv_obj_t * root_page;
static lv_obj_t * create_text(lv_obj_t * parent, const char * icon, const␣
↪char * txt,
                              lv_menu_builder_variant_t builder_variant);
static lv_obj_t * create_slider(lv_obj_t * parent,
                                const char * icon, const char * txt, int32_
↪t min, int32_t max, int32_t val);
static lv_obj_t * create_switch(lv_obj_t * parent,
                                const char * icon, const char * txt, bool␣
↪chk);

void lv_example_menu_5(void)
{
    lv_obj_t * menu = lv_menu_create(lv_screen_active());

    lv_color_t bg_color = lv_obj_get_style_bg_color(menu, LV_PART_MAIN);
    if(lv_color_brightness(bg_color) > 127) {
        lv_obj_set_style_bg_color(menu, lv_color_darken(lv_obj_get_style_
↪bg_color(menu, LV_PART_MAIN), 10), 0);
    }
    else {
        lv_obj_set_style_bg_color(menu, lv_color_darken(lv_obj_get_style_
↪bg_color(menu, LV_PART_MAIN), 50), 0);
    }
    lv_menu_set_mode_root_back_button(menu, LV_MENU_ROOT_BACK_BUTTON_
```

(continues on next page)

(continua dalla pagina precedente)

```
↪ENABLED);
    lv_obj_add_event_cb(menu, back_event_handler, LV_EVENT_CLICKED, menu);
    lv_obj_set_size(menu, lv_display_get_horizontal_resolution(NULL), lv_
↪display_get_vertical_resolution(NULL));
    lv_obj_center(menu);

    lv_obj_t * cont;
    lv_obj_t * section;

    /*Create sub pages*/
    lv_obj_t * sub_mechanics_page = lv_menu_page_create(menu, NULL);
    lv_obj_set_style_pad_hor(sub_mechanics_page, lv_obj_get_style_pad_
↪left(lv_menu_get_main_header(menu), LV_PART_MAIN), 0);
    lv_menu_separator_create(sub_mechanics_page);
    section = lv_menu_section_create(sub_mechanics_page);
    create_slider(section, LV_SYMBOL_SETTINGS, "Velocity", 0, 150, 120);
    create_slider(section, LV_SYMBOL_SETTINGS, "Acceleration", 0, 150, 50);
    create_slider(section, LV_SYMBOL_SETTINGS, "Weight limit", 0, 150, 80);

    lv_obj_t * sub_sound_page = lv_menu_page_create(menu, NULL);
    lv_obj_set_style_pad_hor(sub_sound_page, lv_obj_get_style_pad_left(lv_
↪menu_get_main_header(menu), LV_PART_MAIN), 0);
    lv_menu_separator_create(sub_sound_page);
    section = lv_menu_section_create(sub_sound_page);
    create_switch(section, LV_SYMBOL_AUDIO, "Sound", false);

    lv_obj_t * sub_display_page = lv_menu_page_create(menu, NULL);
    lv_obj_set_style_pad_hor(sub_display_page, lv_obj_get_style_pad_
↪left(lv_menu_get_main_header(menu), LV_PART_MAIN), 0);
    lv_menu_separator_create(sub_display_page);
    section = lv_menu_section_create(sub_display_page);
    create_slider(section, LV_SYMBOL_SETTINGS, "Brightness", 0, 150, 100);

    lv_obj_t * sub_software_info_page = lv_menu_page_create(menu, NULL);
    lv_obj_set_style_pad_hor(sub_software_info_page, lv_obj_get_style_pad_
↪left(lv_menu_get_main_header(menu), LV_PART_MAIN),
                             0);
    section = lv_menu_section_create(sub_software_info_page);
    create_text(section, NULL, "Version 1.0", LV_MENU_ITEM_BUILDER_VARIANT_
↪1);

    lv_obj_t * sub_legal_info_page = lv_menu_page_create(menu, NULL);
    lv_obj_set_style_pad_hor(sub_legal_info_page, lv_obj_get_style_pad_
↪left(lv_menu_get_main_header(menu), LV_PART_MAIN),
                             0);
    section = lv_menu_section_create(sub_legal_info_page);
    for(uint32_t i = 0; i < 15; i++) {
        create_text(section, NULL,
                    "This is a long long long long long long long long␣
↪long text, if it is long enough it may scroll.",
                    LV_MENU_ITEM_BUILDER_VARIANT_1);
    }

    lv_obj_t * sub_about_page = lv_menu_page_create(menu, NULL);
    lv_obj_set_style_pad_hor(sub_about_page, lv_obj_get_style_pad_left(lv_
↪menu_get_main_header(menu), LV_PART_MAIN), 0);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_menu_separator_create(sub_about_page);
    section = lv_menu_section_create(sub_about_page);
    cont = create_text(section, NULL, "Software information", LV_MENU_ITEM_
↪BUILDER_VARIANT_1);
    lv_menu_set_load_page_event(menu, cont, sub_software_info_page);
    cont = create_text(section, NULL, "Legal information", LV_MENU_ITEM_
↪BUILDER_VARIANT_1);
    lv_menu_set_load_page_event(menu, cont, sub_legal_info_page);

    lv_obj_t * sub_menu_mode_page = lv_menu_page_create(menu, NULL);
    lv_obj_set_style_pad_hor(sub_menu_mode_page, lv_obj_get_style_pad_
↪left(lv_menu_get_main_header(menu), LV_PART_MAIN), 0);
    lv_menu_separator_create(sub_menu_mode_page);
    section = lv_menu_section_create(sub_menu_mode_page);
    cont = create_switch(section, LV_SYMBOL_AUDIO, "Sidebar enable", true);
    lv_obj_add_event_cb(lv_obj_get_child(cont, 2), switch_handler, LV_
↪EVENT_VALUE_CHANGED, menu);

    /*Create a root page*/
    root_page = lv_menu_page_create(menu, "Settings");
    lv_obj_set_style_pad_hor(root_page, lv_obj_get_style_pad_left(lv_menu_
↪get_main_header(menu), LV_PART_MAIN), 0);
    section = lv_menu_section_create(root_page);
    cont = create_text(section, LV_SYMBOL_SETTINGS, "Mechanics", LV_MENU_
↪ITEM_BUILDER_VARIANT_1);
    lv_menu_set_load_page_event(menu, cont, sub_mechanics_page);
    cont = create_text(section, LV_SYMBOL_AUDIO, "Sound", LV_MENU_ITEM_
↪BUILDER_VARIANT_1);
    lv_menu_set_load_page_event(menu, cont, sub_sound_page);
    cont = create_text(section, LV_SYMBOL_SETTINGS, "Display", LV_MENU_
↪ITEM_BUILDER_VARIANT_1);
    lv_menu_set_load_page_event(menu, cont, sub_display_page);

    create_text(root_page, NULL, "Others", LV_MENU_ITEM_BUILDER_VARIANT_1);
    section = lv_menu_section_create(root_page);
    cont = create_text(section, NULL, "About", LV_MENU_ITEM_BUILDER_
↪VARIANT_1);
    lv_menu_set_load_page_event(menu, cont, sub_about_page);
    cont = create_text(section, LV_SYMBOL_SETTINGS, "Menu mode", LV_MENU_
↪ITEM_BUILDER_VARIANT_1);
    lv_menu_set_load_page_event(menu, cont, sub_menu_mode_page);

    lv_menu_set_sidebar_page(menu, root_page);

    lv_obj_send_event(lv_obj_get_child(lv_obj_get_child(lv_menu_get_cur_
↪sidebar_page(menu), 0), 0), LV_EVENT_CLICKED,
                      NULL);
}

static void back_event_handler(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    lv_obj_t * menu = (lv_obj_t *)lv_event_get_user_data(e);

    if(lv_menu_back_button_is_root(menu, obj)) {
        lv_obj_t * mbox1 = lv_msgbox_create(NULL);
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_msgbox_add_title(mbox1, "Hello");
        lv_msgbox_add_text(mbox1, "Root back btn click.");
        lv_msgbox_add_close_button(mbox1);
    }
}

static void switch_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * menu = (lv_obj_t *)lv_event_get_user_data(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        if(lv_obj_has_state(obj, LV_STATE_CHECKED)) {
            lv_menu_set_page(menu, NULL);
            lv_menu_set_sidebar_page(menu, root_page);
            lv_obj_send_event(lv_obj_get_child(lv_obj_get_child(lv_menu_
↪get_cur_sidebar_page(menu), 0), 0), LV_EVENT_CLICKED,
                              NULL);
        }
        else {
            lv_menu_set_sidebar_page(menu, NULL);
            lv_menu_clear_history(menu); /* Clear history because we will␣
↪be showing the root page later */
            lv_menu_set_page(menu, root_page);
        }
    }
}

static lv_obj_t * create_text(lv_obj_t * parent, const char * icon, const␣
↪char * txt,
                              lv_menu_builder_variant_t builder_variant)
{
    lv_obj_t * obj = lv_menu_cont_create(parent);

    lv_obj_t * img = NULL;
    lv_obj_t * label = NULL;

    if(icon) {
        img = lv_image_create(obj);
        lv_image_set_src(img, icon);
    }

    if(txt) {
        label = lv_label_create(obj);
        lv_label_set_text(label, txt);
        lv_label_set_long_mode(label, LV_LABEL_LONG_MODE_SCROLL_CIRCULAR);
        lv_obj_set_flex_grow(label, 1);
    }

    if(builder_variant == LV_MENU_ITEM_BUILDER_VARIANT_2 && icon && txt) {
        lv_obj_add_flag(img, LV_OBJ_FLAG_FLEX_IN_NEW_TRACK);
        lv_obj_swap(img, label);
    }

    return obj;
}
```

(continues on next page)

(continua dalla pagina precedente)

```
static lv_obj_t * create_slider(lv_obj_t * parent, const char * icon,␣
↪const char * txt, int32_t min, int32_t max,
                                int32_t val)
{
    lv_obj_t * obj = create_text(parent, icon, txt, LV_MENU_ITEM_BUILDER_
↪VARIANT_2);

    lv_obj_t * slider = lv_slider_create(obj);
    lv_obj_set_flex_grow(slider, 1);
    lv_slider_set_range(slider, min, max);
    lv_slider_set_value(slider, val, LV_ANIM_OFF);

    if(icon == NULL) {
        lv_obj_add_flag(slider, LV_OBJ_FLAG_FLEX_IN_NEW_TRACK);
    }

    return obj;
}

static lv_obj_t * create_switch(lv_obj_t * parent, const char * icon,␣
↪const char * txt, bool chk)
{
    lv_obj_t * obj = create_text(parent, icon, txt, LV_MENU_ITEM_BUILDER_
↪VARIANT_1);

    lv_obj_t * sw = lv_switch_create(obj);
    lv_obj_add_state(sw, chk ? LV_STATE_CHECKED : LV_STATE_DEFAULT);

    return obj;
}

#endif
```

3.7.21 Casella Messaggi

Casella messaggi semplice

```
#include "../../lv_examples.h"
#if LV_USE_MSGBOX && LV_BUILD_EXAMPLES

static void event_cb(lv_event_t * e)
{
    lv_obj_t * btn = lv_event_get_target_obj(e);
    lv_obj_t * label = lv_obj_get_child(btn, 0);
    LV_UNUSED(label);
    LV_LOG_USER("Button %s clicked", lv_label_get_text(label));
}

void lv_example_msgbox_1(void)
{
    lv_obj_t * mbox1 = lv_msgbox_create(NULL);

    lv_msgbox_add_title(mbox1, "Hello");

    lv_msgbox_add_text(mbox1, "This is a message box with two buttons.");
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_msgbox_add_close_button(mbox1);

    lv_obj_t * btn;
    btn = lv_msgbox_add_footer_button(mbox1, "Apply");
    lv_obj_add_event_cb(btn, event_cb, LV_EVENT_CLICKED, NULL);
    btn = lv_msgbox_add_footer_button(mbox1, "Cancel");
    lv_obj_add_event_cb(btn, event_cb, LV_EVENT_CLICKED, NULL);
    return;
}

#endif
```

Casella messaggi con scorrimento e stile

```
#include "../../lv_examples.h"
#if LV_USE_MSGBOX && LV_BUILD_EXAMPLES

static void minimize_button_event_cb(lv_event_t * e)
{
    lv_obj_t * mbox = (lv_obj_t *) lv_event_get_user_data(e);
    lv_obj_add_flag(mbox, LV_OBJ_FLAG_HIDDEN);
}

void lv_example_msgbox_2(void)
{
    lv_obj_t * setting = lv_msgbox_create(lv_screen_active());
    lv_obj_set_style_clip_corner(setting, true, 0);

    /* setting fixed size */
    lv_obj_set_size(setting, 300, 200);

    /* setting's titlebar/header */
    lv_msgbox_add_title(setting, "Setting");
    lv_obj_t * minimize_button = lv_msgbox_add_header_button(setting, LV_
↪SYMBOL_MINUS);
    lv_obj_add_event_cb(minimize_button, minimize_button_event_cb, LV_
↪EVENT_CLICKED, setting);
    lv_msgbox_add_close_button(setting);

    /* setting's content*/
    lv_obj_t * content = lv_msgbox_get_content(setting);
    lv_obj_set_flex_flow(content, LV_FLEX_FLOW_COLUMN);
    lv_obj_set_flex_align(content, LV_FLEX_ALIGN_START, LV_FLEX_ALIGN_
↪CENTER, LV_FLEX_ALIGN_CENTER);
    lv_obj_set_style_pad_right(content, -1, LV_PART_SCROLLBAR);

    lv_obj_t * cont_brightness = lv_obj_create(content);
    lv_obj_set_size(cont_brightness, lv_pct(100), LV_SIZE_CONTENT);
    lv_obj_set_flex_flow(cont_brightness, LV_FLEX_FLOW_COLUMN);
    lv_obj_set_flex_align(cont_brightness, LV_FLEX_ALIGN_CENTER,  LV_FLEX_
↪ALIGN_START, LV_FLEX_ALIGN_CENTER);

    lv_obj_t * lb_brightness = lv_label_create(cont_brightness);
    lv_label_set_text(lb_brightness, "Brightness : ");
    lv_obj_t * slider_brightness = lv_slider_create(cont_brightness);
    lv_obj_set_width(slider_brightness, lv_pct(100));
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_slider_set_value(slider_brightness, 50, LV_ANIM_OFF);

    lv_obj_t * cont_speed = lv_obj_create(content);
    lv_obj_set_size(cont_speed, lv_pct(100), LV_SIZE_CONTENT);
    lv_obj_set_flex_flow(cont_speed, LV_FLEX_FLOW_COLUMN);
    lv_obj_set_flex_align(cont_speed, LV_FLEX_ALIGN_CENTER,  LV_FLEX_ALIGN_
↪START, LV_FLEX_ALIGN_CENTER);

    lv_obj_t * lb_speed = lv_label_create(cont_speed);
    lv_label_set_text(lb_speed, "Speed : ");
    lv_obj_t * slider_speed = lv_slider_create(cont_speed);
    lv_obj_set_width(slider_speed, lv_pct(100));
    lv_slider_set_value(slider_speed, 80, LV_ANIM_OFF);

    /* footer */
    lv_obj_t * apply_button = lv_msgbox_add_footer_button(setting, "Apply
↪");
    lv_obj_set_flex_grow(apply_button, 1);

    lv_obj_t * cancel_button = lv_msgbox_add_footer_button(setting, "Cancel
↪");
    lv_obj_set_flex_grow(cancel_button, 1);

    lv_obj_t * footer = lv_msgbox_get_footer(setting);
    lv_obj_set_style_bg_color(footer, lv_palette_main(LV_PALETTE_INDIGO),␣
↪0);
    lv_obj_set_style_bg_opa(footer, LV_OPA_100, 0);
}
#endif
```

3.7.22 Roller

Roller semplice

```
#include "../../lv_examples.h"
#if LV_USE_ROLLER && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        char buf[32];
        lv_roller_get_selected_str(obj, buf, sizeof(buf));
        LV_LOG_USER("Selected month: %s\n", buf);
    }
}

/**
 * An infinite roller with the name of the months
 */
void lv_example_roller_1(void)
{
    lv_obj_t * roller1 = lv_roller_create(lv_screen_active());
    lv_roller_set_options(roller1,
                          "January\n"
```

(continues on next page)

(continua dalla pagina precedente)

```
                            "February\n"
                            "March\n"
                            "April\n"
                            "May\n"
                            "June\n"
                            "July\n"
                            "August\n"
                            "September\n"
                            "October\n"
                            "November\n"
                            "December",
                            LV_ROLLER_MODE_INFINITE);

    lv_roller_set_visible_row_count(roller1, 4);
    lv_obj_center(roller1);
    lv_obj_add_event_cb(roller1, event_handler, LV_EVENT_ALL, NULL);
}

#endif
```

Roller con stili

```
#include "../../lv_examples.h"
#if LV_USE_ROLLER && LV_FONT_MONTSERRAT_22 && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        char buf[32];
        lv_roller_get_selected_str(obj, buf, sizeof(buf));
        LV_LOG_USER("Selected value: %s", buf);
    }
}

/**
 * Roller with various alignments and larger text in the selected area
 */
void lv_example_roller_2(void)
{
    /*A style to make the selected option larger*/
    static lv_style_t style_sel;
    lv_style_init(&style_sel);
    lv_style_set_text_font(&style_sel, &lv_font_montserrat_22);
    lv_style_set_bg_color(&style_sel, lv_color_hex3(0xf88));
    lv_style_set_border_width(&style_sel, 2);
    lv_style_set_border_color(&style_sel, lv_color_hex3(0xf00));

    const char * opts = "1\n2\n3\n4\n5\n6\n7\n8\n9\n10";
    lv_obj_t * roller;

    /*A roller on the left with left aligned text, and custom width*/
    roller = lv_roller_create(lv_screen_active());
    lv_roller_set_options(roller, opts, LV_ROLLER_MODE_NORMAL);
    lv_roller_set_visible_row_count(roller, 2);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_width(roller, 100);
    lv_obj_add_style(roller, &style_sel, LV_PART_SELECTED);
    lv_obj_set_style_text_align(roller, LV_TEXT_ALIGN_LEFT, 0);
    lv_obj_set_style_bg_color(roller, lv_color_hex3(0x0f0), 0);
    lv_obj_set_style_bg_grad_color(roller, lv_color_hex3(0xafa), 0);
    lv_obj_set_style_bg_grad_dir(roller, LV_GRAD_DIR_VER, 0);
    lv_obj_align(roller, LV_ALIGN_LEFT_MID, 10, 0);
    lv_obj_add_event_cb(roller, event_handler, LV_EVENT_ALL, NULL);
    lv_roller_set_selected(roller, 2, LV_ANIM_OFF);

    /*A roller on the middle with center aligned text, and auto (default)␣
↪width*/
    roller = lv_roller_create(lv_screen_active());
    lv_roller_set_options(roller, opts, LV_ROLLER_MODE_NORMAL);
    lv_roller_set_visible_row_count(roller, 3);
    lv_obj_add_style(roller, &style_sel, LV_PART_SELECTED);
    lv_obj_align(roller, LV_ALIGN_CENTER, 0, 0);
    lv_obj_add_event_cb(roller, event_handler, LV_EVENT_ALL, NULL);
    lv_roller_set_selected(roller, 5, LV_ANIM_OFF);

    /*A roller on the right with right aligned text, and custom width*/
    roller = lv_roller_create(lv_screen_active());
    lv_roller_set_options(roller, opts, LV_ROLLER_MODE_NORMAL);
    lv_roller_set_visible_row_count(roller, 4);
    lv_obj_set_width(roller, 80);
    lv_obj_add_style(roller, &style_sel, LV_PART_SELECTED);
    lv_obj_set_style_text_align(roller, LV_TEXT_ALIGN_RIGHT, 0);
    lv_obj_align(roller, LV_ALIGN_RIGHT_MID, -10, 0);
    lv_obj_add_event_cb(roller, event_handler, LV_EVENT_ALL, NULL);
    lv_roller_set_selected(roller, 8, LV_ANIM_OFF);
}

#endif
```

Aggiungere una maschera di dissolvenza al roller

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

#if LV_USE_ROLLER && LV_DRAW_SW_COMPLEX && LV_BUILD_EXAMPLES && 1

static void generate_mask(lv_draw_buf_t * mask)
{
    /*Create a "8 bit alpha" canvas and clear it*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, mask);
    lv_canvas_fill_bg(canvas, lv_color_white(), LV_OPA_TRANSP);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    /*Draw a label to the canvas. The result "image" will be used as mask*/
    lv_draw_rect_dsc_t rect_dsc;
    lv_draw_rect_dsc_init(&rect_dsc);
    rect_dsc.bg_grad.dir = LV_GRAD_DIR_VER;
    rect_dsc.bg_grad.stops[0].color = lv_color_black();
    rect_dsc.bg_grad.stops[1].color = lv_color_white();
```

(continues on next page)

(continua dalla pagina precedente)

```
    rect_dsc.bg_grad.stops[0].opa = LV_OPA_COVER;
    rect_dsc.bg_grad.stops[1].opa = LV_OPA_COVER;
    lv_area_t a = {0, 0, mask->header.w - 1, mask->header.h / 2 - 10};
    lv_draw_rect(&layer, &rect_dsc, &a);

    a.y1 = mask->header.h / 2 + 10;
    a.y2 = mask->header.h - 1;
    rect_dsc.bg_grad.stops[0].color = lv_color_white();
    rect_dsc.bg_grad.stops[1].color = lv_color_black();
    lv_draw_rect(&layer, &rect_dsc, &a);

    lv_canvas_finish_layer(canvas, &layer);

    /*Comment it to make the mask visible*/
    lv_obj_delete(canvas);
}

/**
 * Add a fade mask to roller.
 */
void lv_example_roller_3(void)
{
    lv_obj_set_style_bg_color(lv_screen_active(), lv_palette_main(LV_
↪PALETTE_BLUE_GREY), 0);

    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_bg_color(&style, lv_color_black());
    lv_style_set_text_color(&style, lv_color_white());
    lv_style_set_border_width(&style, 0);
    lv_style_set_radius(&style, 0);

    lv_obj_t * roller1 = lv_roller_create(lv_screen_active());
    lv_obj_add_style(roller1, &style, 0);
    lv_obj_set_style_bg_opa(roller1, LV_OPA_50, LV_PART_SELECTED);

    lv_roller_set_options(roller1,
                          "January\n"
                          "February\n"
                          "March\n"
                          "April\n"
                          "May\n"
                          "June\n"
                          "July\n"
                          "August\n"
                          "September\n"
                          "October\n"
                          "November\n"
                          "December",
                          LV_ROLLER_MODE_NORMAL);

    lv_obj_center(roller1);
    lv_roller_set_visible_row_count(roller1, 4);

    /* Create the mask to make the top and bottom part of roller faded.
     * The width and height are empirical values for simplicity*/
```

(continues on next page)

(continua dalla pagina precedente)

```
    LV_DRAW_BUF_DEFINE_STATIC(mask, 130, 150, LV_COLOR_FORMAT_L8);
    LV_DRAW_BUF_INIT_STATIC(mask);

    generate_mask(&mask);
    lv_obj_set_style_bitmap_mask_src(roller1, &mask, 0);
}
#endif
```

3.7.23 Scala

Una semplice scala orizzontale

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES

/**
 * A simple horizontal scale
 */
void lv_example_scale_1(void)
{
    lv_obj_t * scale = lv_scale_create(lv_screen_active());
    lv_obj_set_size(scale, lv_pct(80), 100);
    lv_scale_set_mode(scale, LV_SCALE_MODE_HORIZONTAL_BOTTOM);
    lv_obj_center(scale);

    lv_scale_set_label_show(scale, true);

    lv_scale_set_total_tick_count(scale, 31);
    lv_scale_set_major_tick_every(scale, 5);

    lv_obj_set_style_length(scale, 5, LV_PART_ITEMS);
    lv_obj_set_style_length(scale, 10, LV_PART_INDICATOR);
    lv_scale_set_range(scale, 10, 40);
}

#endif
```

Una scala verticale con sezioni e stile personalizzato

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES

/**
 * An vertical scale with section and custom styling
 */
void lv_example_scale_2(void)
{
    lv_obj_t * scale = lv_scale_create(lv_screen_active());
    lv_obj_set_size(scale, 60, 200);
    lv_scale_set_label_show(scale, true);
    lv_scale_set_mode(scale, LV_SCALE_MODE_VERTICAL_RIGHT);
    lv_obj_center(scale);

    lv_scale_set_total_tick_count(scale, 21);
    lv_scale_set_major_tick_every(scale, 5);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_style_length(scale, 10, LV_PART_INDICATOR);
    lv_obj_set_style_length(scale, 5, LV_PART_ITEMS);
    lv_scale_set_range(scale, 0, 100);

    static const char * custom_labels[] = {"0 °C", "25 °C", "50 °C", "75 °C
↪", "100 °C", NULL};
    lv_scale_set_text_src(scale, custom_labels);

    static lv_style_t indicator_style;
    lv_style_init(&indicator_style);

    /* Label style properties */
    lv_style_set_text_font(&indicator_style, LV_FONT_DEFAULT);
    lv_style_set_text_color(&indicator_style, lv_palette_darken(LV_PALETTE_
↪BLUE, 3));

    /* Major tick properties */
    lv_style_set_line_color(&indicator_style, lv_palette_darken(LV_PALETTE_
↪BLUE, 3));
    lv_style_set_width(&indicator_style, 10U);      /*Tick length*/
    lv_style_set_line_width(&indicator_style, 2U);  /*Tick width*/
    lv_obj_add_style(scale, &indicator_style, LV_PART_INDICATOR);

    static lv_style_t minor_ticks_style;
    lv_style_init(&minor_ticks_style);
    lv_style_set_line_color(&minor_ticks_style, lv_palette_lighten(LV_
↪PALETTE_BLUE, 2));
    lv_style_set_width(&minor_ticks_style, 5U);         /*Tick length*/
    lv_style_set_line_width(&minor_ticks_style, 2U);    /*Tick width*/
    lv_obj_add_style(scale, &minor_ticks_style, LV_PART_ITEMS);

    static lv_style_t main_line_style;
    lv_style_init(&main_line_style);
    /* Main line properties */
    lv_style_set_line_color(&main_line_style, lv_palette_darken(LV_PALETTE_
↪BLUE, 3));
    lv_style_set_line_width(&main_line_style, 2U); // Tick width
    lv_obj_add_style(scale, &main_line_style, LV_PART_MAIN);

    /* Add a section */
    static lv_style_t section_minor_tick_style;
    static lv_style_t section_label_style;
    static lv_style_t section_main_line_style;

    lv_style_init(&section_label_style);
    lv_style_init(&section_minor_tick_style);
    lv_style_init(&section_main_line_style);

    /* Label style properties */
    lv_style_set_text_font(&section_label_style, LV_FONT_DEFAULT);
    lv_style_set_text_color(&section_label_style, lv_palette_darken(LV_
↪PALETTE_RED, 3));

    lv_style_set_line_color(&section_label_style, lv_palette_darken(LV_
↪PALETTE_RED, 3));
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_set_line_width(&section_label_style, 5U); /*Tick width*/

    lv_style_set_line_color(&section_minor_tick_style, lv_palette_
↪lighten(LV_PALETTE_RED, 2));
    lv_style_set_line_width(&section_minor_tick_style, 4U); /*Tick width*/

    /* Main line properties */
    lv_style_set_line_color(&section_main_line_style, lv_palette_darken(LV_
↪PALETTE_RED, 3));
    lv_style_set_line_width(&section_main_line_style, 4U); /*Tick width*/

    /* Configure section styles */
    lv_scale_section_t * section = lv_scale_add_section(scale);
    lv_scale_set_section_range(scale, section, 75, 100);
    lv_scale_set_section_style_indicator(scale, section, &section_label_
↪style);
    lv_scale_set_section_style_items(scale, section, &section_minor_tick_
↪style);
    lv_scale_set_section_style_main(scale, section, &section_main_line_
↪style);


    lv_obj_set_style_bg_color(scale, lv_palette_main(LV_PALETTE_BLUE_GREY),
↪ 0);
    lv_obj_set_style_bg_opa(scale, LV_OPA_50, 0);
    lv_obj_set_style_pad_left(scale, 8, 0);
    lv_obj_set_style_radius(scale, 8, 0);
    lv_obj_set_style_pad_ver(scale, 20, 0);
}

#endif
```

Una semplice scala circolare

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES

LV_IMAGE_DECLARE(img_hand);

static lv_obj_t * needle_line;
static lv_obj_t * needle_img;

static void set_needle_line_value(void * obj, int32_t v)
{
    lv_scale_set_line_needle_value((lv_obj_t *)obj, needle_line, 60, v);
}

static void set_needle_img_value(void * obj, int32_t v)
{
    lv_scale_set_image_needle_value((lv_obj_t *)obj, needle_img, v);
}

/**
 * A simple round scale
 */
void lv_example_scale_3(void)
```

(continues on next page)

(continua dalla pagina precedente)

```
{
    lv_obj_t * scale_line = lv_scale_create(lv_screen_active());

    lv_obj_set_size(scale_line, 150, 150);
    lv_scale_set_mode(scale_line, LV_SCALE_MODE_ROUND_INNER);
    lv_obj_set_style_bg_opa(scale_line, LV_OPA_COVER, 0);
    lv_obj_set_style_bg_color(scale_line, lv_palette_lighten(LV_PALETTE_
↪RED, 5), 0);
    lv_obj_set_style_radius(scale_line, LV_RADIUS_CIRCLE, 0);
    lv_obj_set_style_clip_corner(scale_line, true, 0);
    lv_obj_align(scale_line, LV_ALIGN_LEFT_MID, LV_PCT(2), 0);

    lv_scale_set_label_show(scale_line, true);

    lv_scale_set_total_tick_count(scale_line, 31);
    lv_scale_set_major_tick_every(scale_line, 5);

    lv_obj_set_style_length(scale_line, 5, LV_PART_ITEMS);
    lv_obj_set_style_length(scale_line, 10, LV_PART_INDICATOR);
    lv_scale_set_range(scale_line, 10, 40);

    lv_scale_set_angle_range(scale_line, 270);
    lv_scale_set_rotation(scale_line, 135);

    needle_line = lv_line_create(scale_line);

    lv_obj_set_style_line_width(needle_line, 6, LV_PART_MAIN);
    lv_obj_set_style_line_rounded(needle_line, true, LV_PART_MAIN);

    lv_anim_t anim_scale_line;
    lv_anim_init(&anim_scale_line);
    lv_anim_set_var(&anim_scale_line, scale_line);
    lv_anim_set_exec_cb(&anim_scale_line, set_needle_line_value);
    lv_anim_set_duration(&anim_scale_line, 1000);
    lv_anim_set_repeat_count(&anim_scale_line, LV_ANIM_REPEAT_INFINITE);
    lv_anim_set_reverse_duration(&anim_scale_line, 1000);
    lv_anim_set_values(&anim_scale_line, 10, 40);
    lv_anim_start(&anim_scale_line);

    lv_obj_t * scale_img = lv_scale_create(lv_screen_active());

    lv_obj_set_size(scale_img, 150, 150);
    lv_scale_set_mode(scale_img, LV_SCALE_MODE_ROUND_INNER);
    lv_obj_set_style_bg_opa(scale_img, LV_OPA_COVER, 0);
    lv_obj_set_style_bg_color(scale_img, lv_palette_lighten(LV_PALETTE_RED,
↪ 5), 0);
    lv_obj_set_style_radius(scale_img, LV_RADIUS_CIRCLE, 0);
    lv_obj_set_style_clip_corner(scale_img, true, 0);
    lv_obj_align(scale_img, LV_ALIGN_RIGHT_MID, LV_PCT(-2), 0);

    lv_scale_set_label_show(scale_img, true);

    lv_scale_set_total_tick_count(scale_img, 31);
    lv_scale_set_major_tick_every(scale_img, 5);

    lv_obj_set_style_length(scale_img, 5, LV_PART_ITEMS);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_style_length(scale_img, 10, LV_PART_INDICATOR);
    lv_scale_set_range(scale_img, 10, 40);

    lv_scale_set_angle_range(scale_img, 270);
    lv_scale_set_rotation(scale_img, 135);

    /* image must point to the right. E.g. -O------>*/
    needle_img = lv_image_create(scale_img);
    lv_image_set_src(needle_img, &img_hand);
    lv_obj_align(needle_img, LV_ALIGN_CENTER, 47, -2);
    lv_image_set_pivot(needle_img, 3, 4);

    lv_anim_t anim_scale_img;
    lv_anim_init(&anim_scale_img);
    lv_anim_set_var(&anim_scale_img, scale_img);
    lv_anim_set_exec_cb(&anim_scale_img, set_needle_img_value);
    lv_anim_set_duration(&anim_scale_img, 1000);
    lv_anim_set_repeat_count(&anim_scale_img, LV_ANIM_REPEAT_INFINITE);
    lv_anim_set_reverse_duration(&anim_scale_img, 1000);
    lv_anim_set_values(&anim_scale_img, 10, 40);
    lv_anim_start(&anim_scale_img);
}

#endif
```

Una scala circolare con sezione e stile personalizzato

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES

/**
 * A round scale with section and custom styling
 */
void lv_example_scale_4(void)
{
    lv_obj_t * scale = lv_scale_create(lv_screen_active());
    lv_obj_set_size(scale, 150, 150);
    lv_scale_set_label_show(scale, true);
    lv_scale_set_mode(scale, LV_SCALE_MODE_ROUND_OUTER);
    lv_obj_center(scale);

    lv_scale_set_total_tick_count(scale, 21);
    lv_scale_set_major_tick_every(scale, 5);

    lv_obj_set_style_length(scale, 5, LV_PART_ITEMS);
    lv_obj_set_style_length(scale, 10, LV_PART_INDICATOR);
    lv_scale_set_range(scale, 0, 100);

    static const char * custom_labels[] = {"0 °C", "25 °C", "50 °C", "75 °C
↪", "100 °C", NULL};
    lv_scale_set_text_src(scale, custom_labels);

    static lv_style_t indicator_style;
    lv_style_init(&indicator_style);

    /* Label style properties */
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_set_text_font(&indicator_style, LV_FONT_DEFAULT);
    lv_style_set_text_color(&indicator_style, lv_palette_darken(LV_PALETTE_
↪BLUE, 3));

    /* Major tick properties */
    lv_style_set_line_color(&indicator_style, lv_palette_darken(LV_PALETTE_
↪BLUE, 3));
    lv_style_set_width(&indicator_style, 10U);      /*Tick length*/
    lv_style_set_line_width(&indicator_style, 2U);  /*Tick width*/
    lv_obj_add_style(scale, &indicator_style, LV_PART_INDICATOR);

    static lv_style_t minor_ticks_style;
    lv_style_init(&minor_ticks_style);
    lv_style_set_line_color(&minor_ticks_style, lv_palette_lighten(LV_
↪PALETTE_BLUE, 2));
    lv_style_set_width(&minor_ticks_style, 5U);         /*Tick length*/
    lv_style_set_line_width(&minor_ticks_style, 2U);    /*Tick width*/
    lv_obj_add_style(scale, &minor_ticks_style, LV_PART_ITEMS);

    static lv_style_t main_line_style;
    lv_style_init(&main_line_style);
    /* Main line properties */
    lv_style_set_arc_color(&main_line_style, lv_palette_darken(LV_PALETTE_
↪BLUE, 3));
    lv_style_set_arc_width(&main_line_style, 2U); /*Tick width*/
    lv_obj_add_style(scale, &main_line_style, LV_PART_MAIN);

    /* Add a section */
    static lv_style_t section_minor_tick_style;
    static lv_style_t section_label_style;
    static lv_style_t section_main_line_style;

    lv_style_init(&section_label_style);
    lv_style_init(&section_minor_tick_style);
    lv_style_init(&section_main_line_style);

    /* Label style properties */
    lv_style_set_text_font(&section_label_style, LV_FONT_DEFAULT);
    lv_style_set_text_color(&section_label_style, lv_palette_darken(LV_
↪PALETTE_RED, 3));

    lv_style_set_line_color(&section_label_style, lv_palette_darken(LV_
↪PALETTE_RED, 3));
    lv_style_set_line_width(&section_label_style, 5U); /*Tick width*/

    lv_style_set_line_color(&section_minor_tick_style, lv_palette_
↪lighten(LV_PALETTE_RED, 2));
    lv_style_set_line_width(&section_minor_tick_style, 4U); /*Tick width*/

    /* Main line properties */
    lv_style_set_arc_color(&section_main_line_style, lv_palette_darken(LV_
↪PALETTE_RED, 3));
    lv_style_set_arc_width(&section_main_line_style, 4U); /*Tick width*/

    /* Configure section styles */
    lv_scale_section_t * section = lv_scale_add_section(scale);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_scale_set_section_range(scale, section, 75, 100);
    lv_scale_set_section_style_indicator(scale, section, &section_label_
↪style);
    lv_scale_set_section_style_items(scale, section, &section_minor_tick_
↪style);
    lv_scale_set_section_style_main(scale, section, &section_main_line_
↪style);

}

#endif
```

Una scala con sezione e stile personalizzato

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES

/**
 * An scale with section and custom styling
 */
void lv_example_scale_5(void)
{
    lv_obj_t * scale = lv_scale_create(lv_screen_active());
    lv_obj_set_size(scale, lv_display_get_horizontal_resolution(NULL) / 2,␣
↪lv_display_get_vertical_resolution(NULL) / 2);
    lv_scale_set_label_show(scale, true);

    lv_scale_set_total_tick_count(scale, 10);
    lv_scale_set_major_tick_every(scale, 5);

    lv_obj_set_style_length(scale, 5, LV_PART_ITEMS);
    lv_obj_set_style_length(scale, 10, LV_PART_INDICATOR);
    lv_scale_set_range(scale, 25, 35);

    static const char * custom_labels[3] = {"One", "Two", NULL};
    lv_scale_set_text_src(scale, custom_labels);

    static lv_style_t indicator_style;
    lv_style_init(&indicator_style);
    /* Label style properties */
    lv_style_set_text_font(&indicator_style, LV_FONT_DEFAULT);
    lv_style_set_text_color(&indicator_style, lv_color_hex(0xff00ff));
    /* Major tick properties */
    lv_style_set_line_color(&indicator_style, lv_color_hex(0x00ff00));
    lv_style_set_width(&indicator_style, 10U); // Tick length
    lv_style_set_line_width(&indicator_style, 2U); // Tick width
    lv_obj_add_style(scale, &indicator_style, LV_PART_INDICATOR);

    static lv_style_t minor_ticks_style;
    lv_style_init(&minor_ticks_style);
    lv_style_set_line_color(&minor_ticks_style, lv_color_hex(0xff0000));
    lv_style_set_width(&minor_ticks_style, 5U); // Tick length
    lv_style_set_line_width(&minor_ticks_style, 2U); // Tick width
    lv_obj_add_style(scale, &minor_ticks_style, LV_PART_ITEMS);

    static lv_style_t main_line_style;
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_init(&main_line_style);
    /* Main line properties */
    lv_style_set_line_color(&main_line_style, lv_color_hex(0x0000ff));
    lv_style_set_line_width(&main_line_style, 2U); // Tick width
    lv_obj_add_style(scale, &main_line_style, LV_PART_MAIN);

    lv_obj_center(scale);

    /* Add a section */
    static lv_style_t section_minor_tick_style;
    static lv_style_t section_label_style;

    lv_style_init(&section_label_style);
    lv_style_init(&section_minor_tick_style);

    /* Label style properties */
    lv_style_set_text_font(&section_label_style, LV_FONT_DEFAULT);
    lv_style_set_text_color(&section_label_style, lv_color_hex(0xff0000));
    lv_style_set_text_letter_space(&section_label_style, 10);
    lv_style_set_text_opa(&section_label_style, LV_OPA_50);

    lv_style_set_line_color(&section_label_style, lv_color_hex(0xff0000));
    // lv_style_set_width(&section_label_style, 20U); // Tick length
    lv_style_set_line_width(&section_label_style, 5U); // Tick width

    lv_style_set_line_color(&section_minor_tick_style, lv_color_
↪hex(0x0000ff));
    // lv_style_set_width(&section_label_style, 20U); // Tick length
    lv_style_set_line_width(&section_minor_tick_style, 4U); // Tick width

    /* Configure section styles */
    lv_scale_section_t * section = lv_scale_add_section(scale);
    lv_scale_set_section_range(scale, section, 25, 30);
    lv_scale_set_section_style_indicator(scale, section, &section_label_
↪style);
    lv_scale_set_section_style_items(scale, section, &section_minor_tick_
↪style);
}

#endif
```

Una scala circolare con più lancette, simile a un orologio

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES

#if LV_USE_FLOAT
    #define my_PRIprecise "f"
#else
    #define my_PRIprecise LV_PRId32
#endif

static lv_obj_t * scale;
static lv_obj_t * minute_hand;
static lv_obj_t * hour_hand;
static lv_point_precise_t minute_hand_points[2];
```

(continues on next page)

(continua dalla pagina precedente)

```
static int32_t hour;
static int32_t minute;

static void timer_cb(lv_timer_t * timer)
{
    LV_UNUSED(timer);

    minute++;
    if(minute > 59) {
        minute = 0;
        hour++;
        if(hour > 11) {
            hour = 0;
        }
    }

    /**
     * the scale will store the needle line points in the existing
     * point array if one was set with `lv_line_set_points_mutable`.
     * Otherwise, it will allocate the needle line points.
     */

    /* the scale will store the minute hand line points in `minute_hand_
↪points` */
    lv_scale_set_line_needle_value(scale, minute_hand, 60, minute);
    /* log the points that were stored in the array */
    LV_LOG_USER(
        "minute hand points - "
        "0: (%" my_PRIprecise ", %" my_PRIprecise "), "
        "1: (%" my_PRIprecise ", %" my_PRIprecise ")",
        minute_hand_points[0].x, minute_hand_points[0].y,
        minute_hand_points[1].x, minute_hand_points[1].y
    );

    /* the scale will allocate the hour hand line points */
    lv_scale_set_line_needle_value(scale, hour_hand, 40, hour * 5 +␣
↪(minute / 12));
}

/**
 * A round scale with multiple needles, resembling a clock
 */
void lv_example_scale_6(void)
{
    scale = lv_scale_create(lv_screen_active());

    lv_obj_set_size(scale, 150, 150);
    lv_scale_set_mode(scale, LV_SCALE_MODE_ROUND_INNER);
    lv_obj_set_style_bg_opa(scale, LV_OPA_60, 0);
    lv_obj_set_style_bg_color(scale, lv_color_black(), 0);
    lv_obj_set_style_radius(scale, LV_RADIUS_CIRCLE, 0);
    lv_obj_set_style_clip_corner(scale, true, 0);
    lv_obj_center(scale);

    lv_scale_set_label_show(scale, true);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_scale_set_total_tick_count(scale, 61);
    lv_scale_set_major_tick_every(scale, 5);

    static const char * hour_ticks[] = {"12", "1", "2", "3", "4", "5", "6",
↪ "7", "8", "9", "10", "11", NULL};
    lv_scale_set_text_src(scale, hour_ticks);

    static lv_style_t indicator_style;
    lv_style_init(&indicator_style);

    /* Label style properties */
    lv_style_set_text_font(&indicator_style, LV_FONT_DEFAULT);
    lv_style_set_text_color(&indicator_style, lv_palette_main(LV_PALETTE_
↪YELLOW));

    /* Major tick properties */
    lv_style_set_line_color(&indicator_style, lv_palette_main(LV_PALETTE_
↪YELLOW));
    lv_style_set_length(&indicator_style, 8); /* tick length */
    lv_style_set_line_width(&indicator_style, 2); /* tick width */
    lv_obj_add_style(scale, &indicator_style, LV_PART_INDICATOR);

    /* Minor tick properties */
    static lv_style_t minor_ticks_style;
    lv_style_init(&minor_ticks_style);
    lv_style_set_line_color(&minor_ticks_style, lv_palette_main(LV_PALETTE_
↪YELLOW));
    lv_style_set_length(&minor_ticks_style, 6); /* tick length */
    lv_style_set_line_width(&minor_ticks_style, 2); /* tick width */
    lv_obj_add_style(scale, &minor_ticks_style, LV_PART_ITEMS);

    /* Main line properties */
    static lv_style_t main_line_style;
    lv_style_init(&main_line_style);
    lv_style_set_arc_color(&main_line_style, lv_color_black());
    lv_style_set_arc_width(&main_line_style, 5);
    lv_obj_add_style(scale, &main_line_style, LV_PART_MAIN);

    lv_scale_set_range(scale, 0, 60);

    lv_scale_set_angle_range(scale, 360);
    lv_scale_set_rotation(scale, 270);

    minute_hand = lv_line_create(scale);
    lv_line_set_points_mutable(minute_hand, minute_hand_points, 2);

    lv_obj_set_style_line_width(minute_hand, 3, 0);
    lv_obj_set_style_line_rounded(minute_hand, true, 0);
    lv_obj_set_style_line_color(minute_hand, lv_color_white(), 0);

    hour_hand = lv_line_create(scale);

    lv_obj_set_style_line_width(hour_hand, 5, 0);
    lv_obj_set_style_line_rounded(hour_hand, true, 0);
    lv_obj_set_style_line_color(hour_hand, lv_palette_main(LV_PALETTE_RED),
↪ 0);
```

(continues on next page)

(continua dalla pagina precedente)

```
    hour = 11;
    minute = 5;
    lv_timer_t * timer = lv_timer_create(timer_cb, 250, NULL);
    lv_timer_ready(timer);
}

#endif
```

Personalizzazione del colore dell'etichetta principale della scala con l'evento LV_EVENT_DRAW_TASK_ADDED

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES

#include "../../../lvgl_private.h" /*To expose the fields of lv_draw_task_
↪t*/

static void draw_event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);
    lv_draw_label_dsc_t * label_draw_dsc = lv_draw_task_get_label_dsc(draw_
↪task);
    if(base_dsc->part == LV_PART_INDICATOR) {
        if(label_draw_dsc) {
            const lv_color_t color_idx[7] = {
                lv_palette_main(LV_PALETTE_RED),
                lv_palette_main(LV_PALETTE_ORANGE),
                lv_palette_main(LV_PALETTE_YELLOW),
                lv_palette_main(LV_PALETTE_GREEN),
                lv_palette_main(LV_PALETTE_CYAN),
                lv_palette_main(LV_PALETTE_BLUE),
                lv_palette_main(LV_PALETTE_PURPLE),
            };
            uint32_t major_tick = lv_scale_get_major_tick_every(obj);
            label_draw_dsc->color = color_idx[base_dsc->id1 / major_tick];

            /*Free the previously allocated text if needed*/
            if(label_draw_dsc->text_local) lv_free((void *)label_draw_dsc->
↪text);

            /*Malloc the text and set text_local as 1 to make LVGL␣
↪automatically free the text.
             * (Local texts are malloc'd internally by LVGL. Mimic this␣
↪behavior here too)*/
            char tmp_buffer[20] = {0}; /* Big enough buffer */
            lv_snprintf(tmp_buffer, sizeof(tmp_buffer), "%.1f",␣
↪(double)base_dsc->id2);
            label_draw_dsc->text = lv_strdup(tmp_buffer);
            label_draw_dsc->text_local = 1;

            lv_point_t size;
            lv_text_get_size(&size, label_draw_dsc->text, label_draw_dsc->
```

(continues on next page)

(continua dalla pagina precedente)

```
↪font, 0, 0, 1000, LV_TEXT_FLAG_NONE);
            int32_t new_w = size.x;
            int32_t old_w = lv_area_get_width(&draw_task->area);

            /* Distribute the new size equally on both sides */
            draw_task->area.x1 -= (new_w - old_w) / 2;
            draw_task->area.x2 += ((new_w - old_w) + 1) / 2;  /* +1 for␣
↪rounding */

        }
    }
}

/**
 * Customizing scale major tick label color with `LV_EVENT_DRAW_TASK_
↪ADDED` event
 */
void lv_example_scale_7(void)
{
    lv_obj_t * scale = lv_scale_create(lv_screen_active());
    lv_obj_set_size(scale, lv_pct(80), 100);
    lv_scale_set_mode(scale, LV_SCALE_MODE_HORIZONTAL_BOTTOM);
    lv_obj_center(scale);

    lv_scale_set_label_show(scale, true);

    lv_scale_set_total_tick_count(scale, 31);
    lv_scale_set_major_tick_every(scale, 5);

    lv_obj_set_style_length(scale, 5, LV_PART_ITEMS);
    lv_obj_set_style_length(scale, 10, LV_PART_INDICATOR);
    lv_scale_set_range(scale, 10, 40);

    lv_obj_add_event_cb(scale, draw_event_cb, LV_EVENT_DRAW_TASK_ADDED,␣
↪NULL);
    lv_obj_add_flag(scale, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);
}

#endif
```

Una scala circolare con etichette ruotate e traslate

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES


/**
 * A simple round scale with label/tick translation
 */
void lv_example_scale_8(void)
{
    lv_obj_t * scale_line = lv_scale_create(lv_screen_active());
    lv_obj_set_size(scale_line, 150, 150);
    lv_scale_set_mode(scale_line, LV_SCALE_MODE_ROUND_INNER);
    lv_obj_set_style_bg_opa(scale_line, LV_OPA_COVER, 0);
    lv_obj_set_style_bg_color(scale_line, lv_palette_lighten(LV_PALETTE_
```

(continues on next page)

(continua dalla pagina precedente)

```
↪RED, 5), 0);
    lv_obj_set_style_radius(scale_line, LV_RADIUS_CIRCLE, 0);
    lv_obj_align(scale_line, LV_ALIGN_LEFT_MID, LV_PCT(2), 0);

    /*Set the texts' and major ticks' style (make the texts rotated)*/
    lv_obj_set_style_transform_rotation(scale_line, LV_SCALE_LABEL_ROTATE_
↪MATCH_TICKS | LV_SCALE_LABEL_ROTATE_KEEP_UPRIGHT,
                                        LV_PART_INDICATOR);
    lv_obj_set_style_translate_x(scale_line, 10, LV_PART_INDICATOR);
    lv_obj_set_style_length(scale_line, 15, LV_PART_INDICATOR);
    lv_obj_set_style_radial_offset(scale_line, 10, LV_PART_INDICATOR);

    /*Set the style of the minor ticks*/
    lv_obj_set_style_length(scale_line, 10, LV_PART_ITEMS);
    lv_obj_set_style_radial_offset(scale_line, 5, LV_PART_ITEMS);
    lv_obj_set_style_line_opa(scale_line, LV_OPA_50, LV_PART_ITEMS);

    lv_scale_set_label_show(scale_line, true);

    lv_scale_set_total_tick_count(scale_line, 31);
    lv_scale_set_major_tick_every(scale_line, 5);

    lv_scale_set_range(scale_line, 10, 40);

    lv_scale_set_angle_range(scale_line, 270);
    lv_scale_set_rotation(scale_line, 135);

    lv_obj_t * needle_line = lv_line_create(scale_line);

    lv_obj_set_style_line_width(needle_line, 3, LV_PART_MAIN);
    lv_obj_set_style_line_rounded(needle_line, true, LV_PART_MAIN);
    lv_scale_set_line_needle_value(scale_line, needle_line, 60, 33);
}

#endif
```

Una scala orizzontale con etichette ruotate e traslate

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES

/**
 * A simple horizontal scale with transforms
 */
void lv_example_scale_9(void)
{
    lv_obj_t * scale = lv_scale_create(lv_screen_active());
    lv_obj_set_size(scale, 200, 100);
    lv_scale_set_mode(scale, LV_SCALE_MODE_HORIZONTAL_BOTTOM);
    lv_obj_center(scale);

    lv_scale_set_label_show(scale, true);
    lv_obj_set_style_transform_rotation(scale, 450, LV_PART_INDICATOR);
    lv_obj_set_style_length(scale, 30, LV_PART_INDICATOR);
    lv_obj_set_style_translate_x(scale, 5, LV_PART_INDICATOR);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_scale_set_total_tick_count(scale, 31);
    lv_scale_set_major_tick_every(scale, 5);

    lv_obj_set_style_length(scale, 5, LV_PART_ITEMS);
    lv_obj_set_style_length(scale, 10, LV_PART_INDICATOR);
    lv_scale_set_range(scale, 10, 40);
}

#endif
```

Uno stile di scala circolare che simula un cardiofrequenzimetro

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES && LV_FONT_MONTSERRAT_40 && LV_FONT_
↪MONTSERRAT_18

static int32_t hr_value = 98;
static int8_t hr_step = 1;
static lv_obj_t * needle_line = NULL;
static lv_obj_t * hr_value_label = NULL;
static lv_obj_t * bpm_label = NULL;
static lv_obj_t * scale = NULL;

typedef struct {
    lv_style_t items;
    lv_style_t indicator;
    lv_style_t main;
} section_styles_t;

static section_styles_t zone1_styles;
static section_styles_t zone2_styles;
static section_styles_t zone3_styles;
static section_styles_t zone4_styles;
static section_styles_t zone5_styles;

static lv_color_t get_hr_zone_color(int32_t hr)
{
    if(hr < 117) return lv_palette_main(LV_PALETTE_GREY); /* Zone 1 */
    else if(hr < 135) return lv_palette_main(LV_PALETTE_BLUE); /* Zone 2 */
    else if(hr < 158) return lv_palette_main(LV_PALETTE_GREEN); /* Zone 3␣
↪*/
    else if(hr < 176) return lv_palette_main(LV_PALETTE_ORANGE); /* Zone 4␣
↪*/
    else return lv_palette_main(LV_PALETTE_RED); /* Zone 5 */
}

static void hr_anim_timer_cb(lv_timer_t * timer)
{
    LV_UNUSED(timer);

    hr_value += hr_step;

    if(hr_value >= 195) {
        hr_value = 195;
        hr_step = -1;
    }
```

(continues on next page)

(continua dalla pagina precedente)

```
    else if(hr_value <= 98) {
        hr_value = 98;
        hr_step = 1;
    }

    /* Update needle */
    lv_scale_set_line_needle_value(scale, needle_line, -8, hr_value);

    /* Update HR text */
    lv_label_set_text_fmt(hr_value_label, "%d", hr_value);

    /* Update text color based on zone */
    lv_color_t zone_color = get_hr_zone_color(hr_value);
    lv_obj_set_style_text_color(hr_value_label, zone_color, 0);
    lv_obj_set_style_text_color(bpm_label, zone_color, 0);
}

static void init_section_styles(section_styles_t * styles, lv_color_t␣
↪color)
{
    lv_style_init(&styles->items);
    lv_style_set_line_color(&styles->items, color);
    lv_style_set_line_width(&styles->items, 0);

    lv_style_init(&styles->indicator);
    lv_style_set_line_color(&styles->indicator, color);
    lv_style_set_line_width(&styles->indicator, 0);

    lv_style_init(&styles->main);
    lv_style_set_arc_color(&styles->main, color);
    lv_style_set_arc_width(&styles->main, 20);
}

static void add_section(lv_obj_t * target_scale,
                        int32_t from,
                        int32_t to,
                        const section_styles_t * styles)
{
    lv_scale_section_t * sec = lv_scale_add_section(target_scale);
    lv_scale_set_section_range(target_scale, sec, from, to);
    lv_scale_set_section_style_items(target_scale, sec, &styles->items);
    lv_scale_set_section_style_indicator(target_scale, sec, &styles->
↪indicator);
    lv_scale_set_section_style_main(target_scale, sec, &styles->main);
}

void lv_example_scale_10(void)
{
    scale = lv_scale_create(lv_screen_active());
    lv_obj_center(scale);
    lv_obj_set_size(scale, 200, 200);

    lv_scale_set_mode(scale, LV_SCALE_MODE_ROUND_INNER);
    lv_scale_set_range(scale, 98, 195);
    lv_scale_set_total_tick_count(scale, 15);
    lv_scale_set_major_tick_every(scale, 3);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_scale_set_angle_range(scale, 280);
    lv_scale_set_rotation(scale, 130);
    lv_scale_set_label_show(scale, false);

    lv_obj_set_style_length(scale, 6, LV_PART_ITEMS);
    lv_obj_set_style_length(scale, 10, LV_PART_INDICATOR);
    lv_obj_set_style_arc_width(scale, 0, LV_PART_MAIN);

    /* Zone 1: (Grey) */
    init_section_styles(&zone1_styles, lv_palette_main(LV_PALETTE_GREY));
    add_section(scale, 98, 117, &zone1_styles);

    /* Zone 2: (Blue) */
    init_section_styles(&zone2_styles, lv_palette_main(LV_PALETTE_BLUE));
    add_section(scale, 117, 135, &zone2_styles);

    /* Zone 3: (Green) */
    init_section_styles(&zone3_styles, lv_palette_main(LV_PALETTE_GREEN));
    add_section(scale, 135, 158, &zone3_styles);

    /* Zone 4: (Orange) */
    init_section_styles(&zone4_styles, lv_palette_main(LV_PALETTE_ORANGE));
    add_section(scale, 158, 176, &zone4_styles);

    /* Zone 5: (Red) */
    init_section_styles(&zone5_styles, lv_palette_main(LV_PALETTE_RED));
    add_section(scale, 176, 195, &zone5_styles);

    needle_line = lv_line_create(scale);

    /* Optional styling */
    lv_obj_set_style_line_color(needle_line, lv_color_black(), LV_PART_
↪MAIN);
    lv_obj_set_style_line_width(needle_line, 12, LV_PART_MAIN);
    lv_obj_set_style_length(needle_line, 20, LV_PART_MAIN);
    lv_obj_set_style_line_rounded(needle_line, false, LV_PART_MAIN);
    lv_obj_set_style_pad_right(needle_line, 50, LV_PART_MAIN);

    int32_t current_hr = 145;

    lv_scale_set_line_needle_value(scale, needle_line, 0, current_hr);

    lv_obj_t * circle = lv_obj_create(lv_scr_act());
    lv_obj_set_size(circle, 130, 130);
    lv_obj_center(circle);

    lv_obj_set_style_radius(circle, LV_RADIUS_CIRCLE, 0);

    lv_obj_set_style_bg_color(circle, lv_obj_get_style_bg_color(lv_scr_
↪act(), LV_PART_MAIN), 0);
    lv_obj_set_style_bg_opa(circle, LV_OPA_COVER, 0);
    lv_obj_set_style_border_width(circle, 0, LV_PART_MAIN);

    lv_obj_t * hr_container = lv_obj_create(circle);
    lv_obj_center(hr_container);
    lv_obj_set_size(hr_container, lv_pct(100), LV_SIZE_CONTENT);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_style_bg_opa(hr_container, LV_OPA_TRANSP, 0);
    lv_obj_set_style_border_width(hr_container, 0, 0);
    lv_obj_set_layout(hr_container, LV_LAYOUT_FLEX);
    lv_obj_set_flex_flow(hr_container, LV_FLEX_FLOW_COLUMN);
    lv_obj_set_style_pad_all(hr_container, 0, LV_PART_MAIN);
    lv_obj_set_style_pad_row(hr_container, 0, 0);
    lv_obj_set_flex_align(hr_container, LV_FLEX_ALIGN_CENTER, LV_FLEX_
↪ALIGN_CENTER, LV_FLEX_ALIGN_CENTER);

    hr_value_label = lv_label_create(hr_container);
    lv_label_set_text_fmt(hr_value_label, "%d", current_hr);
    lv_obj_set_style_text_font(hr_value_label, &lv_font_montserrat_40, 0);
    lv_obj_set_style_text_align(hr_value_label, LV_TEXT_ALIGN_CENTER, 0);

    bpm_label = lv_label_create(hr_container);
    lv_label_set_text(bpm_label, "bpm");
    lv_obj_set_style_text_font(bpm_label, &lv_font_montserrat_18, 0);
    lv_obj_set_style_text_align(bpm_label, LV_TEXT_ALIGN_CENTER, 0);

    lv_color_t zone_color = get_hr_zone_color(current_hr);
    lv_obj_set_style_text_color(hr_value_label, zone_color, 0);
    lv_obj_set_style_text_color(bpm_label, zone_color, 0);

    lv_timer_create(hr_anim_timer_cb, 80, NULL);
}

#endif
```

Uno stile di scala circolare che simula un widget tramonto/alba

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES && LV_FONT_MONTSERRAT_12 && LV_FONT_
↪MONTSERRAT_14 && LV_FONT_MONTSERRAT_16 && LV_FONT_MONTSERRAT_20

static void label_color_cb(lv_event_t * e)
{
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    if(!draw_task) return;

    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);
    if(!base_dsc || base_dsc->part != LV_PART_INDICATOR) return;

    lv_draw_label_dsc_t * label_dsc = lv_draw_task_get_label_dsc(draw_
↪task);
    if(!label_dsc || !label_dsc->text) return;

    const char * txt = label_dsc->text;

    if(lv_strcmp(txt, "06") == 0 || lv_strcmp(txt, "12") == 0 ||
       lv_strcmp(txt, "18") == 0 || lv_strcmp(txt, "24") == 0) {
        label_dsc->color = lv_color_white();
    }
    else {
        label_dsc->color = lv_palette_darken(LV_PALETTE_GREY, 1);
    }
```

(continues on next page)

(continua dalla pagina precedente)

```
}

void lv_example_scale_11(void)
{
    lv_obj_t * bg = lv_obj_create(lv_screen_active());
    lv_obj_set_size(bg, 210, 210);
    lv_obj_center(bg);
    lv_obj_set_style_radius(bg, LV_RADIUS_CIRCLE, 0);
    lv_obj_set_style_bg_color(bg, lv_palette_darken(LV_PALETTE_GREY, 4),␣
↪0);
    lv_obj_set_style_bg_opa(bg, LV_OPA_COVER, 0);
    lv_obj_remove_flag(bg, LV_OBJ_FLAG_SCROLLABLE);
    lv_obj_set_style_pad_all(bg, 0, LV_PART_MAIN);

    lv_obj_t * scale = lv_scale_create(bg);
    lv_obj_center(scale);
    lv_obj_set_size(scale, 150, 150);
    lv_obj_set_style_arc_width(scale, 5, LV_PART_MAIN);

    lv_scale_set_mode(scale, LV_SCALE_MODE_ROUND_OUTER);
    lv_scale_set_range(scale, 0, 24);
    lv_scale_set_total_tick_count(scale, 25);
    lv_scale_set_major_tick_every(scale, 1);
    lv_scale_set_angle_range(scale, 360);
    lv_scale_set_rotation(scale, 105);
    lv_scale_set_label_show(scale, true);
    lv_obj_set_style_text_font(scale, &lv_font_montserrat_12, LV_PART_
↪INDICATOR);
    lv_obj_set_style_pad_radial(scale, -6, LV_PART_INDICATOR);

    /*Rotate the labels of the ticks*/
    lv_obj_set_style_transform_rotation(scale, LV_SCALE_LABEL_ROTATE_MATCH_
↪TICKS | LV_SCALE_LABEL_ROTATE_KEEP_UPRIGHT,
                                        LV_PART_INDICATOR);

    /* Style for major ticks */
    static lv_style_t style_ticks;
    lv_style_init(&style_ticks);
    lv_style_set_line_color(&style_ticks, lv_palette_darken(LV_PALETTE_
↪GREY, 1));
    lv_style_set_line_width(&style_ticks, 2);
    lv_style_set_width(&style_ticks, 10);
    lv_obj_add_style(scale, &style_ticks, LV_PART_INDICATOR);

    /* Style for NIGHT - blue */
    static lv_style_t style_night;
    lv_style_init(&style_night);
    lv_style_set_arc_color(&style_night, lv_palette_main(LV_PALETTE_BLUE));

    /* Style for DAY - dark yellow */
    static lv_style_t style_day;
    lv_style_init(&style_day);
    lv_style_set_arc_color(&style_day, lv_palette_darken(LV_PALETTE_YELLOW,
↪ 3));

    /* NIGHT section */
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_scale_section_t * section_night1 = lv_scale_add_section(scale);
    lv_scale_set_section_range(scale, section_night1, 17, 5);
    lv_scale_set_section_style_main(scale, section_night1, &style_night);

    /* DAY section */
    lv_scale_section_t * section_day = lv_scale_add_section(scale);
    lv_scale_set_section_range(scale, section_day, 5, 17);
    lv_scale_set_section_style_main(scale, section_day, &style_day);


    static const char * hour_labels[] = {
        "01", "02", "03", "04", "05",
        "06", "07", "08", "09", "10",
        "11", "12", "13", "14", "15",
        "16", "17", "18", "19", "20",
        "21", "22", "23", "24",
        NULL
    };
    lv_scale_set_text_src(scale, hour_labels);

    lv_obj_add_flag(scale, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);
    lv_obj_add_event_cb(scale, label_color_cb, LV_EVENT_DRAW_TASK_ADDED,␣
↪NULL);

    lv_obj_t * today = lv_label_create(bg);
    lv_label_set_text(today, "TODAY");
    lv_obj_set_style_text_font(today, &lv_font_montserrat_16, 0);
    lv_obj_set_style_text_color(today, lv_color_white(), 0);
    lv_obj_align(today, LV_ALIGN_TOP_MID, 0, 60);

    lv_obj_t * sunrise_lbl = lv_label_create(bg);
    lv_label_set_text(sunrise_lbl, "SUNRISE");
    lv_obj_set_style_text_font(sunrise_lbl, &lv_font_montserrat_14, 0);
    lv_obj_set_style_text_color(sunrise_lbl, lv_palette_main(LV_PALETTE_
↪GREY), 0);
    lv_obj_align(sunrise_lbl, LV_ALIGN_LEFT_MID, 37, -10);

    lv_obj_t * sunrise_time = lv_label_create(bg);
    lv_label_set_text(sunrise_time, "6:43");
    lv_obj_set_style_text_font(sunrise_time, &lv_font_montserrat_20, 0);
    lv_obj_set_style_text_color(sunrise_time, lv_color_white(), 0);
    lv_obj_align_to(sunrise_time, sunrise_lbl, LV_ALIGN_OUT_BOTTOM_MID, 0,␣
↪2);

    lv_obj_t * sunset_lbl = lv_label_create(bg);
    lv_label_set_text(sunset_lbl, "SUNSET");
    lv_obj_set_style_text_font(sunset_lbl, &lv_font_montserrat_14, 0);
    lv_obj_set_style_text_color(sunset_lbl, lv_palette_main(LV_PALETTE_
↪GREY), 0);
    lv_obj_align(sunset_lbl, LV_ALIGN_RIGHT_MID, -37, -10);

    lv_obj_t * sunset_time = lv_label_create(bg);
    lv_label_set_text(sunset_time, "17:37");
    lv_obj_set_style_text_font(sunset_time, &lv_font_montserrat_20, 0);
    lv_obj_set_style_text_color(sunset_time, lv_color_white(), 0);
    lv_obj_align_to(sunset_time, sunset_lbl, LV_ALIGN_OUT_BOTTOM_MID, 0,␣
```

(continues on next page)

(continua dalla pagina precedente)

```
↪2);
}

#endif
```

Uno stile di scala circolare che simula una bussola

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES

static lv_obj_t * scale;
static lv_obj_t * label;

static const char * heading_to_cardinal(int32_t heading)
{
    /* Normalize heading to range [0, 360) */
    while(heading < 0) heading += 360;
    while(heading >= 360) heading -= 360;

    if(heading < 23) return "N";
    else if(heading < 68) return "NE";
    else if(heading < 113) return "E";
    else if(heading < 158) return "SE";
    else if(heading < 203) return "S";
    else if(heading < 248) return "SW";
    else if(heading < 293) return "W";
    else if(heading < 338) return "NW";

    return "N";
}

static void set_heading_value(void * obj, int32_t v)
{
    LV_UNUSED(obj);
    lv_scale_set_rotation(scale, 270 - v);
    lv_label_set_text_fmt(label, "%d°\n%s", v, heading_to_cardinal(v));
}

static void draw_event_cb(lv_event_t * e)
{
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);
    lv_draw_label_dsc_t * label_draw_dsc = lv_draw_task_get_label_dsc(draw_
↪task);
    lv_draw_line_dsc_t * line_draw_dsc = lv_draw_task_get_line_dsc(draw_
↪task);
    if(base_dsc->part == LV_PART_INDICATOR) {
        if(label_draw_dsc) {
            if(base_dsc->id1 == 0) {
                label_draw_dsc->color = lv_palette_main(LV_PALETTE_RED);
            }
        }
        if(line_draw_dsc) {
            if(base_dsc->id1 == 60) {
                line_draw_dsc->color = lv_palette_main(LV_PALETTE_RED);
```

(continues on next page)

(continua dalla pagina precedente)

```
            }
        }
    }
}

/**
 * A round scale style simulating a compass
 */
void lv_example_scale_12(void)
{
    scale = lv_scale_create(lv_screen_active());

    lv_obj_set_size(scale, 200, 200);
    lv_scale_set_mode(scale, LV_SCALE_MODE_ROUND_INNER);
    lv_obj_set_align(scale, LV_ALIGN_CENTER);

    lv_scale_set_total_tick_count(scale, 61);
    lv_scale_set_major_tick_every(scale, 5);

    lv_obj_set_style_length(scale, 5, LV_PART_ITEMS);
    lv_obj_set_style_length(scale, 10, LV_PART_INDICATOR);
    lv_obj_set_style_line_width(scale, 3, LV_PART_INDICATOR);
    lv_scale_set_range(scale, 0, 360);

    static const char * custom_labels[] = {"N", "30", "60", "E", "120",
↪"150", "S", "210", "240", "W", "300", "330", NULL};
    lv_scale_set_text_src(scale, custom_labels);

    lv_scale_set_angle_range(scale, 360);
    lv_scale_set_rotation(scale, 270);

    lv_obj_add_event_cb(scale, draw_event_cb, LV_EVENT_DRAW_TASK_ADDED,␣
↪NULL);
    lv_obj_add_flag(scale, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);

    label = lv_label_create(lv_screen_active());
    lv_obj_set_width(label, 100);
    lv_obj_set_align(label, LV_ALIGN_CENTER);
    lv_label_set_text(label, "0°\nN");
    lv_obj_set_style_text_align(label, LV_TEXT_ALIGN_CENTER, 0);

    set_heading_value(NULL, 0);

    lv_obj_t * symbol = lv_label_create(scale);
    lv_obj_set_align(symbol, LV_ALIGN_TOP_MID);
    lv_obj_set_y(symbol, 5);
    lv_label_set_text(symbol, LV_SYMBOL_UP);
    lv_obj_set_style_text_align(symbol, LV_TEXT_ALIGN_CENTER, 0);
    lv_obj_set_style_text_color(symbol, lv_palette_main(LV_PALETTE_RED),␣
↪0);

    lv_anim_t anim_scale;
    lv_anim_init(&anim_scale);
    lv_anim_set_var(&anim_scale, scale);
    lv_anim_set_exec_cb(&anim_scale, set_heading_value);
    lv_anim_set_duration(&anim_scale, 5000);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_anim_set_repeat_delay(&anim_scale, 500);
    lv_anim_set_repeat_count(&anim_scale, LV_ANIM_REPEAT_INFINITE);
    lv_anim_set_reverse_duration(&anim_scale, 5000);
    lv_anim_set_reverse_delay(&anim_scale, 500);
    lv_anim_set_values(&anim_scale, 0, 360);
    lv_anim_start(&anim_scale);

}

#endif
```

3.7.24 Slider

Slider semplice

```
#include "../../lv_examples.h"
#if LV_USE_SLIDER && LV_BUILD_EXAMPLES

static void slider_event_cb(lv_event_t * e);
static lv_obj_t * slider_label;

/**
 * A default slider with a label displaying the current value
 */
void lv_example_slider_1(void)
{
    /*Create a slider in the center of the display*/
    lv_obj_t * slider = lv_slider_create(lv_screen_active());
    lv_obj_center(slider);
    lv_obj_add_event_cb(slider, slider_event_cb, LV_EVENT_VALUE_CHANGED,␣
 ↪NULL);

    lv_obj_set_style_anim_duration(slider, 2000, 0);
    /*Create a label below the slider*/
    slider_label = lv_label_create(lv_screen_active());
    lv_label_set_text(slider_label, "0%");

    lv_obj_align_to(slider_label, slider, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
}

static void slider_event_cb(lv_event_t * e)
{
    lv_obj_t * slider = lv_event_get_target_obj(e);
    char buf[8];
    lv_snprintf(buf, sizeof(buf), "%d%%", (int)lv_slider_get_
 ↪value(slider));
    lv_label_set_text(slider_label, buf);
    lv_obj_align_to(slider_label, slider, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
}

#endif
```

Slider con stile personalizzato

```
#include "../../lv_examples.h"
#if LV_USE_SLIDER && LV_BUILD_EXAMPLES

/**
 * Show how to style a slider.
 */
void lv_example_slider_2(void)
{
    /*Create a transition*/
    static const lv_style_prop_t props[] = {LV_STYLE_BG_COLOR, 0};
    static lv_style_transition_dsc_t transition_dsc;
    lv_style_transition_dsc_init(&transition_dsc, props, lv_anim_path_
↪linear, 300, 0, NULL);

    static lv_style_t style_main;
    static lv_style_t style_indicator;
    static lv_style_t style_knob;
    static lv_style_t style_pressed_color;
    lv_style_init(&style_main);
    lv_style_set_bg_opa(&style_main, LV_OPA_COVER);
    lv_style_set_bg_color(&style_main, lv_color_hex3(0xbbb));
    lv_style_set_radius(&style_main, LV_RADIUS_CIRCLE);
    lv_style_set_pad_ver(&style_main, -2); /*Makes the indicator larger*/

    lv_style_init(&style_indicator);
    lv_style_set_bg_opa(&style_indicator, LV_OPA_COVER);
    lv_style_set_bg_color(&style_indicator, lv_palette_main(LV_PALETTE_
↪CYAN));
    lv_style_set_radius(&style_indicator, LV_RADIUS_CIRCLE);
    lv_style_set_transition(&style_indicator, &transition_dsc);

    lv_style_init(&style_knob);
    lv_style_set_bg_opa(&style_knob, LV_OPA_COVER);
    lv_style_set_bg_color(&style_knob, lv_palette_main(LV_PALETTE_CYAN));
    lv_style_set_border_color(&style_knob, lv_palette_darken(LV_PALETTE_
↪CYAN, 3));
    lv_style_set_border_width(&style_knob, 2);
    lv_style_set_radius(&style_knob, LV_RADIUS_CIRCLE);
    lv_style_set_pad_all(&style_knob, 6); /*Makes the knob larger*/
    lv_style_set_transition(&style_knob, &transition_dsc);

    lv_style_init(&style_pressed_color);
    lv_style_set_bg_color(&style_pressed_color, lv_palette_darken(LV_
↪PALETTE_CYAN, 2));

    /*Create a slider and add the style*/
    lv_obj_t * slider = lv_slider_create(lv_screen_active());
    lv_obj_remove_style_all(slider);        /*Remove the styles coming␣
↪from the theme*/

    lv_obj_add_style(slider, &style_main, LV_PART_MAIN);
    lv_obj_add_style(slider, &style_indicator, LV_PART_INDICATOR);
    lv_obj_add_style(slider, &style_pressed_color, LV_PART_INDICATOR | LV_
↪STATE_PRESSED);
    lv_obj_add_style(slider, &style_knob, LV_PART_KNOB);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_add_style(slider, &style_pressed_color, LV_PART_KNOB | LV_STATE_
↪PRESSED);

    lv_obj_center(slider);
}

#endif
```

Slider estensibile

```
#include "../../lv_examples.h"
#if LV_USE_SLIDER && LV_BUILD_EXAMPLES

#define MAX_VALUE 100
#define MIN_VALUE 0

static void slider_event_cb(lv_event_t * e);

/**
 * Show the current value when the slider is pressed by extending the␣
↪drawer
 *
 */
void lv_example_slider_3(void)
{
    /*Create a slider in the center of the display*/
    lv_obj_t * slider;
    slider = lv_slider_create(lv_screen_active());
    lv_obj_center(slider);

    lv_slider_set_mode(slider, LV_SLIDER_MODE_RANGE);
    lv_slider_set_range(slider, MIN_VALUE, MAX_VALUE);
    lv_slider_set_value(slider, 70, LV_ANIM_OFF);
    lv_slider_set_start_value(slider, 20, LV_ANIM_OFF);

    lv_obj_add_event_cb(slider, slider_event_cb, LV_EVENT_ALL, NULL);
    lv_obj_refresh_ext_draw_size(slider);
}

static void slider_event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);

    /*Provide some extra space for the value*/
    if(code == LV_EVENT_REFR_EXT_DRAW_SIZE) {
        lv_event_set_ext_draw_size(e, 50);
    }
    else if(code == LV_EVENT_DRAW_MAIN_END) {
        if(!lv_obj_has_state(obj, LV_STATE_PRESSED)) return;

        lv_area_t slider_area;
        lv_obj_get_coords(obj, &slider_area);
        lv_area_t indic_area = slider_area;
        lv_area_set_width(&indic_area, lv_area_get_width(&slider_area) *␣
↪lv_slider_get_value(obj) / MAX_VALUE);
```

(continues on next page)

(continua dalla pagina precedente)

```
        indic_area.x1 += lv_area_get_width(&slider_area) * lv_slider_get_
↪left_value(obj) / MAX_VALUE;
        char buf[16];
        lv_snprintf(buf, sizeof(buf), "%d - %d", (int)lv_slider_get_left_
↪value(obj), (int)lv_slider_get_value(obj));

        lv_point_t label_size;
        lv_text_get_size(&label_size, buf, LV_FONT_DEFAULT, 0, 0, LV_COORD_
↪MAX, LV_TEXT_FLAG_NONE);
        lv_area_t label_area;
        label_area.x1 = 0;
        label_area.x2 = label_size.x - 1;
        label_area.y1 = 0;
        label_area.y2 = label_size.y - 1;

        lv_area_align(&indic_area, &label_area, LV_ALIGN_OUT_TOP_MID, 0, -
↪10);

        lv_draw_label_dsc_t label_draw_dsc;
        lv_draw_label_dsc_init(&label_draw_dsc);
        label_draw_dsc.color = lv_color_hex3(0x888);
        label_draw_dsc.text = buf;
        label_draw_dsc.text_local = true;
        lv_layer_t * layer = lv_event_get_layer(e);
        lv_draw_label(layer, &label_draw_dsc, &label_area);
    }
}

#endif
```

Slider con direzione opposta

```
#include "../../lv_examples.h"
#if LV_USE_SLIDER && LV_BUILD_EXAMPLES

static void slider_event_cb(lv_event_t * e);
static lv_obj_t * slider_label;

/**
 * Slider with opposite direction
 */
void lv_example_slider_4(void)
{
    /*Create a slider in the center of the display*/
    lv_obj_t * slider = lv_slider_create(lv_screen_active());
    lv_obj_center(slider);
    lv_obj_add_event_cb(slider, slider_event_cb, LV_EVENT_VALUE_CHANGED,␣
↪NULL);
    /*Reverse the direction of the slider*/
    lv_slider_set_range(slider, 100, 0);
    /*Create a label below the slider*/
    slider_label = lv_label_create(lv_screen_active());
    lv_label_set_text(slider_label, "0%");

    lv_obj_align_to(slider_label, slider, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
}
```

(continues on next page)

(continua dalla pagina precedente)

```
static void slider_event_cb(lv_event_t * e)
{
    lv_obj_t * slider = lv_event_get_target_obj(e);
    char buf[8];
    lv_snprintf(buf, sizeof(buf), "%d%%", (int)lv_slider_get_
↪value(slider));
    lv_label_set_text(slider_label, buf);
    lv_obj_align_to(slider_label, slider, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
}

#endif
```

3.7.25 Span

Span con stili personalizzati

```
#include "../../lv_examples.h"
#if LV_USE_SPAN && LV_BUILD_EXAMPLES

static void click_event_cb(lv_event_t * e)
{
    lv_indev_t * indev = lv_event_get_indev(e);
    lv_point_t point;
    lv_indev_get_point(indev, &point);
#if LV_USE_LOG
    lv_obj_t * spans = lv_event_get_target_obj(e);
    lv_span_t * span = lv_spangroup_get_span_by_point(spans, &point);
    LV_LOG_USER("%s", span ? lv_span_get_text(span) : "NULL");
#endif
}

/**
 * Create spans and get clicked one
 */
void lv_example_span_1(void)
{
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_border_width(&style, 1);
    lv_style_set_border_color(&style, lv_palette_main(LV_PALETTE_ORANGE));
    lv_style_set_pad_all(&style, 2);

    lv_obj_t * spans = lv_spangroup_create(lv_screen_active());
    /* Setting a fixed width and height to LV_SIZE_CONTENT will make the␣
↪text wrap */
    lv_obj_set_width(spans, 300);
    lv_obj_set_height(spans, LV_SIZE_CONTENT);
    lv_obj_center(spans);
    lv_obj_add_style(spans, &style, 0);
    lv_obj_add_flag(spans, LV_OBJ_FLAG_CLICKABLE);

    lv_spangroup_set_align(spans, LV_TEXT_ALIGN_LEFT);
    lv_spangroup_set_overflow(spans, LV_SPAN_OVERFLOW_CLIP);
    lv_spangroup_set_indent(spans, 20);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_span_t * span = lv_spangroup_add_span(spans);
    lv_span_set_text(span, "China is a beautiful country.");
    lv_style_set_text_color(lv_span_get_style(span), lv_palette_main(LV_
↪PALETTE_RED));
    lv_style_set_text_decor(lv_span_get_style(span), LV_TEXT_DECOR_
↪UNDERLINE);
    lv_style_set_text_opa(lv_span_get_style(span), LV_OPA_50);

    span = lv_spangroup_add_span(spans);
    lv_span_set_text_static(span, "good good study, day day up.");
#if LV_FONT_MONTSERRAT_24
    lv_style_set_text_font(lv_span_get_style(span),  &lv_font_montserrat_
↪24);
#endif
    lv_style_set_text_color(lv_span_get_style(span), lv_palette_main(LV_
↪PALETTE_GREEN));

    span = lv_spangroup_add_span(spans);
    lv_span_set_text_static(span, "LVGL is an open-source graphics library.
↪");
    lv_style_set_text_color(lv_span_get_style(span), lv_palette_main(LV_
↪PALETTE_BLUE));

    span = lv_spangroup_add_span(spans);
    lv_span_set_text_static(span, "the boy no name.");
    lv_style_set_text_color(lv_span_get_style(span), lv_palette_main(LV_
↪PALETTE_GREEN));
#if LV_FONT_MONTSERRAT_20
    lv_style_set_text_font(lv_span_get_style(span), &lv_font_montserrat_
↪20);
#endif
    lv_style_set_text_decor(lv_span_get_style(span), LV_TEXT_DECOR_
↪UNDERLINE);

    span = lv_spangroup_add_span(spans);
    lv_span_set_text(span, "I have a dream that hope to come true.");
    lv_style_set_text_decor(lv_span_get_style(span), LV_TEXT_DECOR_
↪STRIKETHROUGH);

    lv_spangroup_refresh(spans);

    lv_obj_add_event_cb(spans, click_event_cb, LV_EVENT_CLICKED, NULL);
}

#endif
```

3.7.26 Spinbox

Spinbox semplice

```
#include "../../lv_examples.h"
#if LV_USE_SPINBOX && LV_BUILD_EXAMPLES

static lv_obj_t * spinbox;

static void lv_spinbox_increment_event_cb(lv_event_t * e)
```

(continues on next page)

(continua dalla pagina precedente)

```
{
    lv_event_code_t code = lv_event_get_code(e);
    if(code == LV_EVENT_SHORT_CLICKED || code  == LV_EVENT_LONG_PRESSED_
↪REPEAT) {
        lv_spinbox_increment(spinbox);
    }
}

static void lv_spinbox_decrement_event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    if(code == LV_EVENT_SHORT_CLICKED || code == LV_EVENT_LONG_PRESSED_
↪REPEAT) {
        lv_spinbox_decrement(spinbox);
    }
}

void lv_example_spinbox_1(void)
{
    spinbox = lv_spinbox_create(lv_screen_active());
    lv_spinbox_set_range(spinbox, -1000, 25000);
    lv_spinbox_set_digit_format(spinbox, 5, 2);
    lv_spinbox_step_prev(spinbox);
    lv_obj_set_width(spinbox, 100);
    lv_obj_center(spinbox);

    int32_t h = lv_obj_get_height(spinbox);

    lv_obj_t * btn = lv_button_create(lv_screen_active());
    lv_obj_set_size(btn, h, h);
    lv_obj_align_to(btn, spinbox, LV_ALIGN_OUT_RIGHT_MID, 5, 0);
    lv_obj_set_style_bg_image_src(btn, LV_SYMBOL_PLUS, 0);
    lv_obj_add_event_cb(btn, lv_spinbox_increment_event_cb, LV_EVENT_ALL, ␣
↪NULL);

    btn = lv_button_create(lv_screen_active());
    lv_obj_set_size(btn, h, h);
    lv_obj_align_to(btn, spinbox, LV_ALIGN_OUT_LEFT_MID, -5, 0);
    lv_obj_set_style_bg_image_src(btn, LV_SYMBOL_MINUS, 0);
    lv_obj_add_event_cb(btn, lv_spinbox_decrement_event_cb, LV_EVENT_ALL,␣
↪NULL);
}

#endif
```

3.7.27 Spinner

Spinner semplice

```
#include "../../lv_examples.h"
#if LV_USE_SPINNER && LV_BUILD_EXAMPLES

void lv_example_spinner_1(void)
{
    /*Create a spinner*/
    lv_obj_t * spinner = lv_spinner_create(lv_screen_active());
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_size(spinner, 100, 100);
    lv_obj_center(spinner);
    lv_spinner_set_anim_params(spinner, 10000, 200);
}

#endif
```

3.7.28 Switch

Switch semplice

```
#include "../../lv_examples.h"
#if LV_USE_SWITCH && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        LV_UNUSED(obj);
        LV_LOG_USER("State: %s\n", lv_obj_has_state(obj, LV_STATE_CHECKED)␣
↪? "On" : "Off");
    }
}

void lv_example_switch_1(void)
{
    lv_obj_set_flex_flow(lv_screen_active(), LV_FLEX_FLOW_COLUMN);
    lv_obj_set_flex_align(lv_screen_active(), LV_FLEX_ALIGN_CENTER, LV_
↪FLEX_ALIGN_CENTER, LV_FLEX_ALIGN_CENTER);

    lv_obj_t * sw;

    sw = lv_switch_create(lv_screen_active());
    lv_obj_add_event_cb(sw, event_handler, LV_EVENT_ALL, NULL);
    lv_obj_add_flag(sw, LV_OBJ_FLAG_EVENT_BUBBLE);

    sw = lv_switch_create(lv_screen_active());
    lv_obj_add_state(sw, LV_STATE_CHECKED);
    lv_obj_add_event_cb(sw, event_handler, LV_EVENT_ALL, NULL);

    sw = lv_switch_create(lv_screen_active());
    lv_obj_add_state(sw, LV_STATE_DISABLED);
    lv_obj_add_event_cb(sw, event_handler, LV_EVENT_ALL, NULL);

    sw = lv_switch_create(lv_screen_active());
    lv_obj_add_state(sw, LV_STATE_CHECKED);
    lv_obj_add_state(sw, LV_STATE_DISABLED);
    lv_obj_add_event_cb(sw, event_handler, LV_EVENT_ALL, NULL);
}

#endif
```

Orientamento dello Switch

```
#include "../../lv_examples.h"
#if LV_USE_SWITCH && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        LV_UNUSED(obj);
        LV_LOG_USER("State: %s\n", lv_obj_has_state(obj, LV_STATE_CHECKED)␣
↪? "On" : "Off");
    }
}

void lv_example_switch_2(void)
{
    lv_obj_set_flex_flow(lv_screen_active(), LV_FLEX_FLOW_ROW);
    lv_obj_set_flex_align(lv_screen_active(), LV_FLEX_ALIGN_CENTER, LV_
↪FLEX_ALIGN_CENTER, LV_FLEX_ALIGN_CENTER);

    lv_obj_t * sw;

    sw = lv_switch_create(lv_screen_active());
    lv_obj_set_size(sw, 30, 60);
    lv_obj_add_event_cb(sw, event_handler, LV_EVENT_ALL, NULL);

    sw = lv_switch_create(lv_screen_active());
    lv_obj_set_size(sw, 30, 60);
    lv_switch_set_orientation(sw, LV_SWITCH_ORIENTATION_VERTICAL);
    lv_obj_add_state(sw, LV_STATE_CHECKED);
    lv_obj_add_event_cb(sw, event_handler, LV_EVENT_ALL, NULL);
}

#endif
```

3.7.29 Tabella

Tabella semplice

```
#include "../../lv_examples.h"
#if LV_USE_TABLE && LV_BUILD_EXAMPLES

static void draw_event_cb(lv_event_t * e)
{
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);
    /*If the cells are drawn...*/
    if(base_dsc->part == LV_PART_ITEMS) {
        uint32_t row = base_dsc->id1;
        uint32_t col = base_dsc->id2;

        /*Make the texts in the first cell center aligned*/
        if(row == 0) {
            lv_draw_label_dsc_t * label_draw_dsc = lv_draw_task_get_label_
```

(continues on next page)

(continua dalla pagina precedente)

```
↪dsc(draw_task);
            if(label_draw_dsc) {
                label_draw_dsc->align = LV_TEXT_ALIGN_CENTER;
            }
            lv_draw_fill_dsc_t * fill_draw_dsc = lv_draw_task_get_fill_
↪dsc(draw_task);
            if(fill_draw_dsc) {
                fill_draw_dsc->color = lv_color_mix(lv_palette_main(LV_
↪PALETTE_BLUE), fill_draw_dsc->color, LV_OPA_20);
                fill_draw_dsc->opa = LV_OPA_COVER;
            }
        }
        /*In the first column align the texts to the right*/
        else if(col == 0) {
            lv_draw_label_dsc_t * label_draw_dsc = lv_draw_task_get_label_
↪dsc(draw_task);
            if(label_draw_dsc) {
                label_draw_dsc->align = LV_TEXT_ALIGN_RIGHT;
            }
        }

        /*Make every 2nd row grayish*/
        if((row != 0 && row % 2) == 0) {
            lv_draw_fill_dsc_t * fill_draw_dsc = lv_draw_task_get_fill_
↪dsc(draw_task);
            if(fill_draw_dsc) {
                fill_draw_dsc->color = lv_color_mix(lv_palette_main(LV_
↪PALETTE_GREY), fill_draw_dsc->color, LV_OPA_10);
                fill_draw_dsc->opa = LV_OPA_COVER;
            }
        }
    }
}

void lv_example_table_1(void)
{
    lv_obj_t * table = lv_table_create(lv_screen_active());

    /*Fill the first column*/
    lv_table_set_cell_value(table, 0, 0, "Name");
    lv_table_set_cell_value(table, 1, 0, "Apple");
    lv_table_set_cell_value(table, 2, 0, "Banana");
    lv_table_set_cell_value(table, 3, 0, "Lemon");
    lv_table_set_cell_value(table, 4, 0, "Grape");
    lv_table_set_cell_value(table, 5, 0, "Melon");
    lv_table_set_cell_value(table, 6, 0, "Peach");
    lv_table_set_cell_value(table, 7, 0, "Nuts");

    /*Fill the second column*/
    lv_table_set_cell_value(table, 0, 1, "Price");
    lv_table_set_cell_value(table, 1, 1, "$7");
    lv_table_set_cell_value(table, 2, 1, "$4");
    lv_table_set_cell_value(table, 3, 1, "$6");
    lv_table_set_cell_value(table, 4, 1, "$2");
    lv_table_set_cell_value(table, 5, 1, "$5");
    lv_table_set_cell_value(table, 6, 1, "$1");
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_table_set_cell_value(table, 7, 1, "$9");

    /*Set a smaller height to the table. It'll make it scrollable*/
    lv_obj_set_height(table, 200);
    lv_obj_center(table);

    /*Add an event callback to to apply some custom drawing*/
    lv_obj_add_event_cb(table, draw_event_cb, LV_EVENT_DRAW_TASK_ADDED,␣
↪NULL);
    lv_obj_add_flag(table, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);
}

#endif
```

Elenco leggero da tabella

```
#include "../../lv_examples.h"
#if LV_USE_TABLE && LV_BUILD_EXAMPLES

#define ITEM_CNT 200

static void draw_event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);

    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);
    /*If the cells are drawn...*/
    if(base_dsc->part == LV_PART_ITEMS && lv_draw_task_get_type(draw_task)␣
↪== LV_DRAW_TASK_TYPE_FILL) {
        /*Draw the background*/
        bool chk = lv_table_has_cell_ctrl(obj, base_dsc->id1, 0, LV_TABLE_
↪CELL_CTRL_CUSTOM_1);
        lv_draw_rect_dsc_t rect_dsc;
        lv_draw_rect_dsc_init(&rect_dsc);
        rect_dsc.bg_color = chk ? lv_theme_get_color_primary(obj) : lv_
↪palette_lighten(LV_PALETTE_GREY, 2);
        rect_dsc.radius = LV_RADIUS_CIRCLE;

        lv_area_t sw_area;
        sw_area.x1 = 0;
        sw_area.x2 = 40;
        sw_area.y1 = 0;
        sw_area.y2 = 24;
        lv_area_t draw_task_area;
        lv_draw_task_get_area(draw_task, &draw_task_area);
        lv_area_align(&draw_task_area, &sw_area, LV_ALIGN_RIGHT_MID, -15,␣
↪0);
        lv_draw_rect(base_dsc->layer, &rect_dsc, &sw_area);

        /*Draw the knob*/
        rect_dsc.bg_color = lv_color_white();
        lv_area_t knob_area;
        knob_area.x1 = 0;
        knob_area.x2 = 18;
```

(continues on next page)

(continua dalla pagina precedente)

```
        knob_area.y1 = 0;
        knob_area.y2 = 18;
        if(chk) {
            lv_area_align(&sw_area, &knob_area, LV_ALIGN_RIGHT_MID, -3, 0);
        }
        else {
            lv_area_align(&sw_area, &knob_area, LV_ALIGN_LEFT_MID, 3, 0);
        }
        lv_draw_rect(base_dsc->layer, &rect_dsc, &knob_area);
    }
}

static void change_event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    uint32_t col;
    uint32_t row;
    lv_table_get_selected_cell(obj, &row, &col);
    bool chk = lv_table_has_cell_ctrl(obj, row, 0, LV_TABLE_CELL_CTRL_
↪CUSTOM_1);
    if(chk) lv_table_clear_cell_ctrl(obj, row, 0, LV_TABLE_CELL_CTRL_
↪CUSTOM_1);
    else lv_table_set_cell_ctrl(obj, row, 0, LV_TABLE_CELL_CTRL_CUSTOM_1);
}

/**
 * A very light-weighted list created from table
 */
void lv_example_table_2(void)
{
    /*Measure memory usage*/
    lv_mem_monitor_t mon1;
    lv_mem_monitor(&mon1);

    uint32_t t = lv_tick_get();

    lv_obj_t * table = lv_table_create(lv_screen_active());

    /*Set a smaller height to the table. It'll make it scrollable*/
    lv_obj_set_size(table, LV_SIZE_CONTENT, 200);

    lv_table_set_column_width(table, 0, 150);
    lv_table_set_row_count(table, ITEM_CNT); /*Not required but avoids a␣
↪lot of memory reallocation lv_table_set_set_value*/
    lv_table_set_column_count(table, 1);

    /*Don't make the cell pressed, we will draw something different in the␣
↪event*/
    lv_obj_remove_style(table, NULL, LV_PART_ITEMS | LV_STATE_PRESSED);

    uint32_t i;
    for(i = 0; i < ITEM_CNT; i++) {
        lv_table_set_cell_value_fmt(table, i, 0, "Item %" LV_PRIu32, i +␣
↪1);
    }
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_align(table, LV_ALIGN_CENTER, 0, -20);

    /*Add an event callback to to apply some custom drawing*/
    lv_obj_add_event_cb(table, draw_event_cb, LV_EVENT_DRAW_TASK_ADDED,␣
↪NULL);
    lv_obj_add_event_cb(table, change_event_cb, LV_EVENT_VALUE_CHANGED,␣
↪NULL);
    lv_obj_add_flag(table, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);

    lv_mem_monitor_t mon2;
    lv_mem_monitor(&mon2);

    size_t mem_used = mon1.free_size - mon2.free_size;

    uint32_t elaps = lv_tick_elaps(t);

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text_fmt(label, "%" LV_PRIu32" items were created in %"␣
↪LV_PRIu32" ms\n"
                          "using %zu bytes of memory",
                          (uint32_t)ITEM_CNT, elaps, mem_used);

    lv_obj_align(label, LV_ALIGN_BOTTOM_MID, 0, -10);

}

#endif
```

3.7.30 Tabview

Tabview Semplice

```
#include "../../lv_examples.h"
#if LV_USE_TABVIEW && LV_BUILD_EXAMPLES

void lv_example_tabview_1(void)
{
    /*Create a Tab view object*/
    lv_obj_t * tabview;
    tabview = lv_tabview_create(lv_screen_active());

    /*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);
    lv_label_set_text(label, "This the first tab\n\n"
                      "If the content\n"
                      "of a tab\n"
                      "becomes too\n"
                      "longer\n"
                      "than the\n"
                      "container\n"
                      "then it\n"
```

(continues on next page)

(continua dalla pagina precedente)

```
                        "automatically\n"
                        "becomes\n"
                        "scrollable.\n"
                        "\n"
                        "\n"
                        "\n"
                        "Can you see it?");

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

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

    lv_obj_scroll_to_view_recursive(label, LV_ANIM_ON);

}
#endif
```

Tab a sinistra, con stile e nessuno scorrimento

```
#include "../../lv_examples.h"
#if LV_USE_TABVIEW && LV_BUILD_EXAMPLES

/*A vertical tab view with disabled scrolling and some styling*/
void lv_example_tabview_2(void)
{
    /*Create a Tab view object*/
    lv_obj_t * tabview;
    uint32_t tab_count = 0;
    uint32_t i = 0;

    tabview = lv_tabview_create(lv_screen_active());
    lv_tabview_set_tab_bar_position(tabview, LV_DIR_LEFT);
    lv_tabview_set_tab_bar_size(tabview, 80);

    lv_obj_set_style_bg_color(tabview, lv_palette_lighten(LV_PALETTE_RED,␣
↪2), 0);

    lv_obj_t * tab_buttons = lv_tabview_get_tab_bar(tabview);
    lv_obj_set_style_bg_color(tab_buttons, lv_palette_darken(LV_PALETTE_
↪GREY, 3), 0);
    lv_obj_set_style_text_color(tab_buttons, lv_palette_lighten(LV_PALETTE_
↪GREY, 5), 0);

    /*Add 5 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");
    lv_obj_t * tab4 = lv_tabview_add_tab(tabview, "Tab 4");
    lv_obj_t * tab5 = lv_tabview_add_tab(tabview, "Tab 5");

    tab_count = lv_tabview_get_tab_count(tabview);
    for(i = 0; i < tab_count; i++) {
        lv_obj_t * button = lv_obj_get_child(tab_buttons, i);
        lv_obj_set_style_border_side(button, LV_BORDER_SIDE_RIGHT, LV_PART_
```

(continues on next page)

(continua dalla pagina precedente)

```
↪MAIN | LV_STATE_CHECKED);
    }
    lv_obj_set_style_bg_color(tab2, lv_palette_lighten(LV_PALETTE_AMBER,␣
↪3), 0);
    lv_obj_set_style_bg_opa(tab2, LV_OPA_COVER, 0);

    /*Add content to the tabs*/
    lv_obj_t * label = lv_label_create(tab1);
    lv_label_set_text(label, "First tab");

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

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

    label = lv_label_create(tab4);
    lv_label_set_text(label, "Fourth tab");

    label = lv_label_create(tab5);
    lv_label_set_text(label, "Fifth tab");

    lv_obj_remove_flag(lv_tabview_get_content(tabview), LV_OBJ_FLAG_
↪SCROLLABLE);
}
#endif
```

3.7.31 Area di testo

Area di testo semplice

```
#include "../../lv_examples.h"
#if LV_USE_TEXTAREA && LV_BUILD_EXAMPLES

static void textarea_event_handler(lv_event_t * e)
{
    lv_obj_t * ta = lv_event_get_target_obj(e);
    LV_UNUSED(ta);
    LV_LOG_USER("Enter was pressed. The current text is: %s", lv_textarea_
↪get_text(ta));
}

static void btnm_event_handler(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    lv_obj_t * ta = (lv_obj_t *)lv_event_get_user_data(e);
    const char * txt = lv_buttonmatrix_get_button_text(obj, lv_
↪buttonmatrix_get_selected_button(obj));

    if(lv_strcmp(txt, LV_SYMBOL_BACKSPACE) == 0) lv_textarea_delete_
↪char(ta);
    else if(lv_strcmp(txt, LV_SYMBOL_NEW_LINE) == 0) lv_obj_send_event(ta,␣
↪LV_EVENT_READY, NULL);
    else lv_textarea_add_text(ta, txt);

}
```

(continues on next page)

(continua dalla pagina precedente)

```
void lv_example_textarea_1(void)
{
    lv_obj_t * ta = lv_textarea_create(lv_screen_active());
    lv_textarea_set_one_line(ta, true);
    lv_obj_align(ta, LV_ALIGN_TOP_MID, 0, 10);
    lv_obj_add_event_cb(ta, textarea_event_handler, LV_EVENT_READY, ta);
    lv_obj_add_state(ta, LV_STATE_FOCUSED); /*To be sure the cursor is␣
↪visible*/

    static const char * btnm_map[] = {"1", "2", "3", "\n",
                                      "4", "5", "6", "\n",
                                      "7", "8", "9", "\n",
                                      LV_SYMBOL_BACKSPACE, "0", LV_SYMBOL_
↪NEW_LINE, ""
                                     };

    lv_obj_t * btnm = lv_buttonmatrix_create(lv_screen_active());
    lv_obj_set_size(btnm, 200, 150);
    lv_obj_align(btnm, LV_ALIGN_BOTTOM_MID, 0, -10);
    lv_obj_add_event_cb(btnm, btnm_event_handler, LV_EVENT_VALUE_CHANGED,␣
↪ta);
    lv_obj_remove_flag(btnm, LV_OBJ_FLAG_CLICK_FOCUSABLE); /*To keep the␣
↪text area focused on button clicks*/
    lv_buttonmatrix_set_map(btnm, btnm_map);
}

#endif
```

Area di testo con campo password

```
#include "../../lv_examples.h"
#if LV_USE_TEXTAREA && LV_USE_KEYBOARD && LV_BUILD_EXAMPLES

static void ta_event_cb(lv_event_t * e);

static lv_obj_t * kb;

void lv_example_textarea_2(void)
{
    /*Create the password box*/
    lv_obj_t * pwd_ta = lv_textarea_create(lv_screen_active());
    lv_textarea_set_text(pwd_ta, "");
    lv_textarea_set_password_mode(pwd_ta, true);
    lv_textarea_set_one_line(pwd_ta, true);
    lv_obj_set_width(pwd_ta, lv_pct(40));
    lv_obj_set_pos(pwd_ta, 5, 20);
    lv_obj_add_event_cb(pwd_ta, ta_event_cb, LV_EVENT_ALL, NULL);

    /*Create a label and position it above the text box*/
    lv_obj_t * pwd_label = lv_label_create(lv_screen_active());
    lv_label_set_text(pwd_label, "Password:");
    lv_obj_align_to(pwd_label, pwd_ta, LV_ALIGN_OUT_TOP_LEFT, 0, 0);

    /*Create the one-line mode text area*/
    lv_obj_t * text_ta = lv_textarea_create(lv_screen_active());
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_textarea_set_one_line(text_ta, true);
    lv_textarea_set_password_mode(text_ta, false);
    lv_obj_set_width(text_ta, lv_pct(40));
    lv_obj_add_event_cb(text_ta, ta_event_cb, LV_EVENT_ALL, NULL);
    lv_obj_align(text_ta, LV_ALIGN_TOP_RIGHT, -5, 20);

    /*Create a label and position it above the text box*/
    lv_obj_t * oneline_label = lv_label_create(lv_screen_active());
    lv_label_set_text(oneline_label, "Text:");
    lv_obj_align_to(oneline_label, text_ta, LV_ALIGN_OUT_TOP_LEFT, 0, 0);

    /*Create a keyboard*/
    kb = lv_keyboard_create(lv_screen_active());
    lv_obj_set_size(kb,  LV_HOR_RES, LV_VER_RES / 2);

    lv_keyboard_set_textarea(kb, pwd_ta); /*Focus it on one of the text␣
↪areas to start*/

    /*The keyboard will show Arabic characters if they are enabled */
#if LV_USE_ARABIC_PERSIAN_CHARS && LV_FONT_DEJAVU_16_PERSIAN_HEBREW
    lv_obj_set_style_text_font(kb, &lv_font_dejavu_16_persian_hebrew, 0);
    lv_obj_set_style_text_font(text_ta, &lv_font_dejavu_16_persian_hebrew,␣
↪0);
    lv_obj_set_style_text_font(pwd_ta, &lv_font_dejavu_16_persian_hebrew,␣
↪0);
#endif

}

static void ta_event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * ta = lv_event_get_target_obj(e);
    if(code == LV_EVENT_CLICKED || code == LV_EVENT_FOCUSED) {
        /*Focus on the clicked text area*/
        if(kb != NULL) lv_keyboard_set_textarea(kb, ta);
    }

    else if(code == LV_EVENT_READY) {
        LV_LOG_USER("Ready, current text: %s", lv_textarea_get_text(ta));
    }
}

#endif
```

Formattazione automatica del testo

```
#include "../../lv_examples.h"
#if LV_USE_TEXTAREA && LV_USE_KEYBOARD && LV_BUILD_EXAMPLES

static void ta_event_cb(lv_event_t * e);

static lv_obj_t * kb;

/**
 * Automatically format text like a clock. E.g. "12:34"
```

(continues on next page)

(continua dalla pagina precedente)

```
 * Add the ':' automatically.
 */
void lv_example_textarea_3(void)
{
    /*Create the text area*/
    lv_obj_t * ta = lv_textarea_create(lv_screen_active());
    lv_obj_add_event_cb(ta, ta_event_cb, LV_EVENT_VALUE_CHANGED, NULL);
    lv_textarea_set_accepted_chars(ta, "0123456789:");
    lv_textarea_set_max_length(ta, 5);
    lv_textarea_set_one_line(ta, true);
    lv_textarea_set_text(ta, "");

    /*Create a keyboard*/
    kb = lv_keyboard_create(lv_screen_active());
    lv_obj_set_size(kb,  LV_HOR_RES, LV_VER_RES / 2);
    lv_keyboard_set_mode(kb, LV_KEYBOARD_MODE_NUMBER);
    lv_keyboard_set_textarea(kb, ta);
}

static void ta_event_cb(lv_event_t * e)
{
    lv_obj_t * ta = lv_event_get_target_obj(e);
    const char * txt = lv_textarea_get_text(ta);
    if(txt[0] >= '0' && txt[0] <= '9' &&
       txt[1] >= '0' && txt[1] <= '9' &&
       txt[2] != ':') {
        lv_textarea_set_cursor_pos(ta, 2);
        lv_textarea_add_char(ta, ':');
    }
}

#endif
```

Stile del cursore dell'area di testo

```
#include "../../lv_examples.h"
#if LV_USE_TEXTAREA && LV_USE_KEYBOARD && LV_BUILD_EXAMPLES

static void create_styled_textarea_cursor(const char * txt, lv_coord_t y_
↪ofs, lv_style_t * cursor_style)
{
    lv_obj_t * ta = lv_textarea_create(lv_screen_active());
    lv_textarea_set_text(ta, txt);
    lv_obj_set_width(ta, 280);
    lv_obj_align(ta, LV_ALIGN_TOP_MID, 0, y_ofs);
    lv_textarea_set_one_line(ta, true);
    lv_obj_add_state(ta, LV_STATE_FOCUSED);
    lv_obj_add_style(ta, cursor_style, LV_PART_CURSOR | LV_STATE_FOCUSED);
    lv_textarea_set_cursor_pos(ta, 0);
}

void lv_example_textarea_4(void)
{
    static lv_style_t style_simple, style_block, style_underline;

    /* Thin left bar cursor (simple) */
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_init(&style_simple);
    lv_style_set_border_color(&style_simple, lv_palette_main(LV_PALETTE_
↪RED));

    /* Underline cursor */
    lv_style_init(&style_underline);
    lv_style_set_bg_opa(&style_underline, LV_OPA_TRANSP);
    lv_style_set_border_color(&style_underline, lv_palette_main(LV_PALETTE_
↪BLUE));
    lv_style_set_border_side(&style_underline, LV_BORDER_SIDE_BOTTOM);
    lv_style_set_pad_hor(&style_underline, 1); /* set width of cursor␣
↪using pad */
    lv_style_set_border_width(&style_underline, 3); /* set thickness of␣
↪underline cursor */

    /* Full block cursor with many styles */
    lv_style_init(&style_block);
    lv_style_set_bg_opa(&style_block, LV_OPA_COVER);
    lv_style_set_bg_color(&style_block, lv_palette_main(LV_PALETTE_
↪ORANGE));
    lv_style_set_bg_grad_color(&style_block, lv_palette_main(LV_PALETTE_
↪YELLOW));
    lv_style_set_bg_grad_dir(&style_block, LV_GRAD_DIR_VER);
    lv_style_set_border_color(&style_block, lv_palette_main(LV_PALETTE_
↪RED));
    lv_style_set_border_side(&style_block, LV_BORDER_SIDE_FULL);
    lv_style_set_border_width(&style_block, 1);
    lv_style_set_radius(&style_block, 4);
    lv_style_set_text_color(&style_block, lv_color_white());
    lv_style_set_pad_all(&style_block, 1); /* set width of cursor using␣
↪pad */

    /* Create 3 independent textareas, each with a unique styled cursor */
    create_styled_textarea_cursor("This is a simple red cursor", 10,  &
↪style_simple);
    create_styled_textarea_cursor("This is an underline blue cursor", 110,␣
↪&style_underline);
    create_styled_textarea_cursor("This is a complex block cursor", 60, &
↪style_block);
}

#endif
```

3.7.32 Tileview

Tileview con contenuto

```
#include "../../lv_examples.h"
#if LV_USE_TILEVIEW && LV_BUILD_EXAMPLES

/**
 * Create a 2x2 tile view and allow scrolling only in an "L" shape.
 * Demonstrate scroll chaining with a long list that
 * scrolls the tile view when it can't be scrolled further.
 */
void lv_example_tileview_1(void)
```

(continues on next page)

(continua dalla pagina precedente)

```
{
    lv_obj_t * tv = lv_tileview_create(lv_screen_active());

    /*Tile1: just a label*/
    lv_obj_t * tile1 = lv_tileview_add_tile(tv, 0, 0, LV_DIR_BOTTOM);
    lv_obj_t * label = lv_label_create(tile1);
    lv_label_set_text(label, "Scroll down");
    lv_obj_center(label);

    /*Tile2: a button*/
    lv_obj_t * tile2 = lv_tileview_add_tile(tv, 0, 1, (lv_dir_t)(LV_DIR_
↪TOP | LV_DIR_RIGHT));

    lv_obj_t * btn = lv_button_create(tile2);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Scroll up or right");

    lv_obj_set_size(btn, LV_SIZE_CONTENT, LV_SIZE_CONTENT);
    lv_obj_center(btn);

    /*Tile3: a list*/
    lv_obj_t * tile3 = lv_tileview_add_tile(tv, 1, 1, LV_DIR_LEFT);
    lv_obj_t * list = lv_list_create(tile3);
    lv_obj_set_size(list, LV_PCT(100), LV_PCT(100));

    lv_list_add_button(list, NULL, "One");
    lv_list_add_button(list, NULL, "Two");
    lv_list_add_button(list, NULL, "Three");
    lv_list_add_button(list, NULL, "Four");
    lv_list_add_button(list, NULL, "Five");
    lv_list_add_button(list, NULL, "Six");
    lv_list_add_button(list, NULL, "Seven");
    lv_list_add_button(list, NULL, "Eight");
    lv_list_add_button(list, NULL, "Nine");
    lv_list_add_button(list, NULL, "Ten");

}

#endif
```

3.7.33 Window

Window semplice

```
#include "../../lv_examples.h"
#if LV_USE_WIN && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    LV_UNUSED(obj);
    LV_LOG_USER("Button %d clicked", (int)lv_obj_get_index(obj));
}

void lv_example_win_1(void)
```

(continues on next page)

(continua dalla pagina precedente)

```
{
    lv_obj_t * win = lv_win_create(lv_screen_active());
    lv_obj_t * btn;
    btn = lv_win_add_button(win, LV_SYMBOL_LEFT, 40);
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);

    lv_win_add_title(win, "A title");

    btn = lv_win_add_button(win, LV_SYMBOL_RIGHT, 40);
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);

    btn = lv_win_add_button(win, LV_SYMBOL_CLOSE, 60);
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);

    lv_obj_t * cont = lv_win_get_content(win);  /*Content can be added␣
↪here*/
    lv_obj_t * label = lv_label_create(cont);
    lv_label_set_text(label, "This is\n"
                      "a pretty\n"
                      "long text\n"
                      "to see how\n"
                      "the window\n"
                      "becomes\n"
                      "scrollable.\n"
                      "\n"
                      "\n"
                      "Some more\n"
                      "text to be\n"
                      "sure it\n"
                      "overflows. :)");
}

#endif
```

CAPITOLO 4

Integrazione e Driver

4.1 Aggiungere LVGL al Proprio Progetto

4.1.1 Ottenere LVGL

LVGL è disponibile su GitHub: https://github.com/lvgl/lvgl.

Lo si può clonare o eseguire il Download dell'ultima versione della libreria da GitHub.

La libreria grafica vera e propria si trova nella directory `lvgl`. Contiene diverse directory, ma per usare LVGL sono necessari solo i file `.c` e `.h` nella directory `src`, più `lvgl/lvgl.h`, e `lvgl/lv_version.h`.

Demo ed Esempi

La directory `lvgl` contiene anche le directory `examples` e `demos`. Se il progetto necessita di esempi e/o demo, aggiungere queste directory al progetto. Se `make` o *CMake* gestiscono le directory examples e demos, non è richiesta alcuna azione aggiuntiva.

4.1.2 Compilazione di LVGL

Make e CMake

LVGL supporta anche i sistemi di compilazione `make` e `CMake` pronti all'uso. Per aggiungere LVGL al sistema di compilazione basato su Makefile, si aggiungono queste righe al Makefile principale:

```
LVGL_DIR_NAME ?= lvgl     #The name of the lvgl folder (change this if you␣
↪have renamed it)
LVGL_DIR ?= ${shell pwd}  #The path where the lvgl folder is
include $(LVGL_DIR)/$(LVGL_DIR_NAME)/lvgl.mk
```

Per l'integrazione con CMake, consultare questa sezione della *Documentazione*.

Build gestite

TODO

4.1.3 Configurazione

lv_conf.h

Creazione di lv_conf.h

Quando si imposta il progetto per la prima volta, si deve copiare `lvgl/lv_conf_template.h` in `lv_conf.h` accanto alla cartella `lvgl`. Si modifica il primo `#if 0` to `1` per abilitare il contenuto del file e si imposta la definizione *LV_COLOR_DEPTH* in modo che si allinei alla profondità di colore utilizzata dal tuo pannello di visualizzazione. Per i dettagli vedere i commenti in `lv_conf.h`.

Il layout dei file dovrebbe apparire così:

```
lvgl/
lv_conf.h
other files and folders in your project
```

In alternativa, `lv_conf.h` può essere copiato in un'altra posizione, ma in tal caso è necessario aggiungere la definizione `LV_CONF_INCLUDE_SIMPLE` alle opzioni del compilatore (ad esempio `-DLV_CONF_INCLUDE_SIMPLE` per il compilatore GCC) e impostare manualmente il path dell'include (ad esempio `-I../include/gui`). In questo caso, LVGL tenterà di includere `lv_conf.h` semplicemente con `#include "lv_conf.h"`.

È anche possibile utilizzare un nome diverso per `lv_conf.h`. Il path personalizzato può essere impostato tramite la definizione `LV_CONF_PATH`. Per esempio `-DLV_CONF_PATH="/home/joe/my_project/my_custom_conf.h"`. Se questa definizione è settata, `LV_CONF_SKIP` si presume sia `0`. Si noti che, quando si definisce `LV_CONF_PATH`, è necessario assicurarsi che sia definito come una stringa, altrimenti verrà generato un errore di compilazione.

Se `LV_CONF_SKIP` è definito, LVGL non tenterà di includere `lv_conf.h`. Invece, è possibile passare le definizioni di configurazione utilizzando le opzioni di compilazione. Ad esempio `"-DLV_COLOR_DEPTH=32 -DLV_USE_BUTTON=1"`. Le opzioni non impostate otterranno un valore predefinito che è lo stesso del contenuto di `lv_conf_template.h`.

Impostazioni di Configurazione

Una volta installato il file `lv_conf.h` , è possibile modificare questo header per configurare il comportamento di LVGL, disabilitare moduli e funzionalità inutilizzati, regolare la dimensione dei buffer, ecc.

I commenti in `lv_conf.h` spiegano il significato di ciascuna impostazione. Si deve impostare almeno *LV_COLOR_DEPTH* in base alla profondità di colore del display. Nota: gli esempi e le demo devono essere esplicitamente abilitati in `lv_conf.h` se sono necessarie.

TODO: Aggiungere tutto ciò che riguarda il file `lv_conf.h` e il suo contenuto.

Istanze Multiple di LVGL

È possibile eseguire più istanze indipendenti di LVGL. Per abilitare la funzionalità multi-istanza, impostare `LV_GLOBAL_CUSTOM` in `lv_conf.h` e fornire una funzione personalizzata a `lv_global_default()` utilizzando `__thread` o `pthread_key_t`. Questo consentirà l'esecuzione di più istanze di LVGL memorizzando le variabili globali di LVGL in TLS (Thread-Local Storage).

Ad esempio:

```
lv_global_t * lv_global_default(void)
{
    static __thread lv_global_t lv_global;
    return &lv_global;
}
```

Kconfig

LVGL può essere configurato anche tramite Kconfig. Per ora, questa funzionalità è disponibile solo tramite cmake. Internamente, utilizza la porta Python di Kconfig `kconfiglib` per poterlo utilizzare su diverse piattaforme. `kconfiglib` offre l'API Python e alcuni comandi CLI. Ecco un elenco di alcuni comandi utili:

- `menuconfig`: Apre un'interfaccia a menù della console per modificare i valori di configurazione.
- `guiconfig` (richiede `tkinter`): Apre un'interfaccia grafica per modificare i valori di configurazione.
- `savedefconfig`: Salva il file .config corrente come defconfig, elencando solo i valori non di default.
- `alldefconfig`: Crea un file .config con tutti i valori di default.
- `genconfig`: Genera un header C dalla configurazione, seguendo il formato `autoconf.h`.

Prerequisiti

Installare i prerequisiti usando `scripts/install_prerequisites.sh/bat`.

Creazione della configurazione (.config)

A questo punto, il comando `menuconfig` dovrebbe essere disponibile:

```
cd <lvgl_repo>
menuconfig
```

Apportare le modifiche alla configurazione e uscire premendo `Esc` o `Q`, quindi salvare la configurazione. Il file `.config` è stato creato e contiene l'elenco dei valori di configurazione.

Configurazione con cmake

Una volta creato il file``.config``, eseguire cmake con il flag `-DLV_USE_KCONFIG=ON`:

```
cd <lvgl_repo>
cmake -B build -DLV_USE_KCONFIG=ON
cmake --build build
```

Per utilizzare un file `defconfig`, è possibile utilizzare il flag `-DLV_DEFCONFIG_PATH=<path_to_defconfig>`:

```
cd <lvgl_repo>
cmake -B build -DLV_USE_KCONFIG=ON -DLV_DEFCONFIG_PATH=<path_to_defconfig>
cmake --build build
```

Alcuni file defconfig sono disponibili nella cartella `configs/defconfigs`.

Salvataggio di un defconfig

È possibile salvare un defconfig utilizzando il comando `savedefconfig`:

```
cd <lvgl_repo>
menuconfig # make your changes to the default config
savedefconfig
cp defconfig configs/defconfigs/my_custom_defconfig # save it where you␣
↪want
# Then use it to build LVGL
cmake -B build -DLV_USE_KCONFIG=ON -DLV_DEFCONFIG_PATH=configs/defconfigs/
↪my_custom_defconfig
cmake --build build
```

4.1.4 Collegamento di LVGL all'Hardware

Inizializzazione di LVGL

Dopo aver:

- *acquisito LVGL*,
- aggiunto i file LVGL appropriati al progetto e
- *creato un file lv_conf.h* per il progetto,

saranno necessari alcuni passaggi aggiuntivi per avviare il progetto con LVGL.

1. Inizializzare LVGL una volta all'inizio dell'esecuzione del sistema chiamando `lv_init()`. Questa operazione deve essere eseguita prima di qualsiasi altra chiamata a LVGL.
2. Inizializzare i driver.
3. Collegare *Tick Interface*.
4. Collegare *Interfaccia al Display*.
5. Collegare *Interfaccia al Device di Input*.
6. Gestire le attività LVGL relative al tempo chiamando `lv_timer_handler()` ogni pochi millisecondi per gestire i timer LVGL. Consultare *Gestione del Timer* per diversi modi per farlo.
7. Facoltativamente, si imposta un tema con `lv_display_set_theme()`.
8. Dopodiché #include "lvgl/lvgl.h" nei file sorgente ovunque ci sia bisogno di utilizzare le funzioni LVGL.

Tick Interface

LVGL ha bisogno di conoscere l'ora esatta (ovvero il tempo trascorso in millisecondi) per tutte le attività in cui il tempo è un fattore determinante: aggiornamento dei display, lettura dell'input dell'utente, attivazione di eventi, animazioni, ecc.

Esistono due modi per fornire queste informazioni a LVGL:

1. Fornire a LVGL una funzione di callback per recuperare i millisecondi di sistema trascorsi chiamando `lv_tick_set_cb(my_get_milliseconds)`. `my_get_milliseconds()` deve restituire il numero di millisecondi trascorsi dall'avvio del sistema. Molte piattaforme dispongono di funzioni native utilizzabili così come sono. Ad esempio:
   - SDL: `lv_tick_set_cb(SDL_GetTicks);`
   - Arduino: `lv_tick_set_cb(my_tick_get_cb);`, dove `my_tick_get_cb` è: `static uint32_t my_tick_get_cb(void) { return millis(); }`
   - FreeRTOS: `lv_tick_set_cb(xTaskGetTickCount);`
   - STM32: `lv_tick_set_cb(HAL_GetTick);`
   - ESP32: `lv_tick_set_cb(my_tick_get_cb);`, dove `my_tick_get_cb` è un wrapper per `esp_timer_get_time() / 1000;`
2. Chiamare periodicamente `lv_tick_inc(x)`, dove `x` rappresenta i millisecondi trascorsi dall'ultima chiamata. Se `lv_tick_inc()` viene chiamato da un ISR, dovrebbe provenire da un interrupt ad alta priorità o da un interrupt che non può essere perso quando il sistema è sotto carico elevato.

> **Nota**
>
> `lv_tick_inc()` è solo una delle due funzioni LVGL che possono essere chiamate da un interrupt se la scrittura su un valore `uint32_t` è atomica sulla piattaforma. Vedere di seguito la sezione *Considerazioni sui Thread* per saperne di più.

In entrambi i casi, la scrittura del valore Tick `uint32_t` dev'essere *atomic*, il che è solitamente il caso di una piattaforma a 32 o 64 bit. Se si utilizza un sistema a 16 bit (per cui l'aggiornamento del valore Tick non è atomico)

e la piattaforma utilizza il set di istruzioni Harvard, è possibile impostare una funzione come questa come callback passata a `lv_tick_set_cb(my_get_milliseconds)`:

```
/**
 * @brief  Safe read from 'elapsed_power_on_time_in_ms'
 */
uint32_t  my_get_milliseconds()
{
    register uint32_t  u32result;
    /* Disable priority 1-6 interrupts for 2 Fcys. */
    __builtin_disi(2);
    u32result = elapsed_power_on_time_in_ms;   /* Cost: 2 Fcys */
        /* Generally looks like this in assembly:
         *     mov   elapsed_power_on_time_in_ms, W0
         *     mov   0x7898, W1
         * requiring exactly 2 clock cycles.
         * Now value is copied to register pair W0:W1
         * where it can be written to any destination. */
    return u32result;
}
```

Affidabilità

L'avanzamento del valore di tick dovrebbe essere eseguito in modo tale che la sua temporizzazione sia affidabile e non dipenda da nulla che consumi una quantità di tempo sconosciuta. Un esempio di cosa *non* fare: questo può "sembrare" funzionare, ma la temporizzazione di LVGL sarà errata perché il tempo di esecuzione di `lv_timer_handler()` varia da chiamata a chiamata e quindi il ritardo che introduce non può essere noto.

```
// Bad idea
lv_timer_handler();
lv_tick_inc(5);
my_delay_ms(5);
```

Interfaccia al Display

LVGL deve essere dotato di informazioni su ciascun pannello di visualizzazione che si desidera utilizzare. In particolare:

- il formato e la dimensione dei pixel (*Creating a Display*),
- dove renderizzare i pixel (*Draw Buffer(s)*), e
- come inviargli tali pixel renderizzati (*Flush Callback*).

Consultare i rispettivi link per informazioni su come fornire queste informazioni a LVGL.

Interfaccia al Device di Input

LVGL deve sapere come ricevere input da tutti i dispositivi di input utente che verranno utilizzati nel progetto. LVGL supporta un'ampia varietà di dispositivi di input utente:

- touch-screen,
- touch-pad,
- mouse,
- crown [scorrimenti],
- encoder,
- keypad,

- keyboard,
- ecc.

Consultare *Creating an Input Device* per vedere come fare.

API

lv_tick.h

4.1.5 Gestione del Timer

Per gestire i timer di LVGL è necessario chiamare periodicamente `lv_timer_handler()` in uno dei seguenti modi:

- *while(1)* della funzione *main()*, o
- periodicamente, un task del sistema operativo. (Vedere *LVGL e i Thread*.)

Esempio:

```
while(1) {
  uint32_t time_till_next = lv_timer_handler();
  if(time_till_next == LV_NO_TIMER_READY) time_till_next = LV_DEF_REFR_
↪PERIOD; /*handle LV_NO_TIMER_READY. Another option is to `sleep` for␣
↪longer*/
  my_delay_ms(time_till_next);
}
```

Per utilizzare `lv_timer_handler()` in un super-loop, è disponibile la funzione helper `lv_timer_handler_run_in_period()` per semplificare l'impostazione di LVGL con la gestione del tempo:

```
while(1) {
   ...
   lv_timer_handler_run_in_period(5); /* run lv_timer_handler() every 5ms␣
↪*/
   ...
}
```

Oppure utilizzare il tempo di sleep calcolato automaticamente da LVGL:

```
while(1) {
  ...
  lv_timer_periodic_handler();
  ...
}
```

In un ambiente di sistema operativo, è possibile utilizzarlo insieme a **delay** o a **sleep** fornito dal sistema operativo per rilasciare la CPU quando possibile:

```
while (1) {
   uint32_t time_till_next = lv_timer_handler();
   if(time_till_next == LV_NO_TIMER_READY) time_till_next = LV_DEF_REFR_
↪PERIOD; /*handle LV_NO_TIMER_READY. Another option is to `sleep` for␣
↪longer*/
   os_delay_ms(time_till_next); /* delay to avoid unnecessary polling */
}
```

Leggere la sezione *Timer (lv_timer)* per saperne di più sui timer.

Quando Nessun Timer è Pronto

`lv_timer_handler()` restituirà *LV_NO_TIMER_READY* (`UINT32_MAX`) se non ci sono timer in esecuzione. Questo può accadere se non ci sono indev o se sono disabilitati con `lv_indev_enable()`, animazioni in esecuzione o timer creati dall'utente in esecuzione. `lv_timer_handler()` continuerà a restituire *LV_NO_TIMER_READY* finché non c'è un timer in esecuzione. I timer del display rimarranno in pausa quando non c'è motivo di aggiornarli. `lv_timer_handler()` dovrebbe essere chiamato dopo la creazione, l'eliminazione o la modifica di qualcosa in modo che venga eseguito un aggiornamento se necessario. In pratica, questo significa attendere senza timeout un evento esterno. Dopo che gli eventi esterni sono stati ricevuti e gestiti, `lv_timer_handler()` dovrebbe essere chiamato i nuovo.

```
while (1) {
   uint32_t time_till_next = lv_timer_handler();
   int timeout;

   /* Wait forever for events upon LV_NO_TIMER_READY, because there
    * is no reason to call lv_timer_handler sooner. */
   if(time_till_next == LV_NO_TIMER_READY) timeout = -1; /*infinite␣
↪timeout*/

   /*Otherwise, wait for events at least until the timeout expires.*/
   else timeout = time_till_next;

   if(poll(..., timeout)) {
      /*Handle events before calling `lv_timer_handler` again.*/
   }
}
```

Se non c'è una sorgente di eventi esterni, si può scegliere di uscire dal ciclo o semplicemente ritardare a lungo.

Se un altro *thread* sta chiamando funzioni LVGL, si potrebbe voler chiamare nuovamente `lv_timer_handler()` molto presto per gestire gli effetti di quegli altri thread.

API

lv_timer.h

4.1.6 Considerazioni sui Thread

Definizioni

Thread

Nel contesto di questo documento, un thread è una qualsiasi sequenza di istruzioni della CPU. Nelle implementazioni "bare-metal" (ovvero senza sistema operativo), i thread includono:

- il thread principale che esegue un ciclo while(1) che esegue il sistema e
- Le interrupt service routine (ISR).

Quando vengono eseguiti su un sistema operativo, i thread includono:

- ogni task (o processo),
- ISR, e
- i sistemi operativi avanzati possono avere più "thread in esecuzione" all'interno di un processo.

Operazione Atomica

Se l'operazione X è atomica, significa che qualsiasi thread che la osserva la vedrà come non ancora iniziata o come completata, e non in uno stato parzialmente completato.

Se altri thread possono vedere l'operazione in uno stato parzialmente eseguito o interferire con essa, allora l'operazione X non è atomica.

Se un'operazione atomica può fallire, la sua implementazione deve riportare la risorsa allo stato precedente all'avvio dell'operazione. Agli altri thread, deve apparire come se l'operazione non fosse ancora iniziata.

Dati Atomici

Un dato (ovvero il contenuto di una variabile o di una struttura dati) è atomico se qualsiasi thread che lo osserva lo vedrà sempre in uno stato coerente, come se le operazioni su di esso non fossero ancora iniziate o fossero state completate con successo, e non in uno stato parzialmente modificato o altrimenti incoerente.

Quando la lettura o la scrittura di un valore viene avviata e completata con 1 istruzione CPU, è automaticamente atomica, poiché non può mai essere vista in uno stato incoerente (parzialmente modificato), nemmeno a causa di un'interruzione o di un'eccezione CPU. Con tali valori, i programmatori non richiedono alcuna protezione speciale per garantire che tutti i thread li vedano in uno stato coerente.

LVGL e i Thread

LVGL **non è thread-safe**.

Ciò significa che è responsabilità del programmatore assicurarsi che nessuna funzione LVGL venga chiamata mentre è in corso un'altra chiamata LVGL in un altro thread. Questo include le chiamate a `lv_timer_handler()`.

> **Nota**
>
> Supponendo che quanto sopra sia il caso, è sicuro chiamare le funzioni LVGL nelle
>
> - *callback di eventi* e nelle
> - *callback del timer*
>
> perché il thread che gestisce entrambi è il thread che chiama `lv_timer_handler()`.

Motivo:

LVGL gestisce molte strutture dati complesse, che sono "risorse di sistema" che devono essere protette dall'essere "viste" da altri thread in uno stato incoerente. Un'alta percentuale di funzioni LVGL (funzioni che iniziano con `lv_`) leggono o modificano tali strutture dati. Quelle che le modificano pongono i dati in uno stato incoerente durante l'esecuzione (poiché tali modifiche sono sequenze multi-step), ma li riportano a uno stato coerente prima che tali funzioni terminino. Per questo motivo, l'esecuzione di ciascuna funzione LVGL deve essere completata prima che venga avviata qualsiasi altra funzione LVGL.

Eccezioni a quanto Sopra

Queste due funzioni LVGL possono essere chiamate da qualsiasi thread:

- `lv_tick_inc()` (se la scrittura su `uint32_t` è atomica sulla piattaforma; vedere *Tick Interface* per maggiori informazioni) e
- `lv_display_flush_ready()` (*Flush Callback* per maggiori informazioni)

Il motivo per cui questo è accettabile è che i dati LVGL da loro modificati sono *atomici*.

Se un interrupt DEVE trasmettere informazioni a una parte dell'applicazione che chiama funzioni LVGL, imposta un flag o un altro valore atomico che il thread che chiama LVGL (o un *Timer LVGL* che si crea) possa leggere ed eseguire azioni.

Se si sta utilizzando un sistema operativo, ci sono alcune altre opzioni. Vedi di seguito.

Garantire che gli Aggiornamenti Temporali Siano Atomici

Affinché i task LVGL relativi al tempo siano affidabili, gli aggiornamenti temporali tramite l'interfaccia Tick devono essere affidabili e il valore di Tick deve apparire *atomico* a LVGL. *Tick Interface* per i dettagli.

Task

In un sistema operativo, è comune avere molti thread di esecuzione ("task" in alcuni sistemi operativi) che eseguono servizi per l'applicazione. In alcuni casi, tali thread possono acquisire dati che dovrebbero essere visualizzati (o altrimenti riflessi) nell'interfaccia utente, e ciò richiede chiamate LVGL per visualizzare tali dati (o modifiche).

Tuttavia, rimane responsabilità del programmatore assicurarsi che nessuna funzione LVGL venga chiamata mentre è in corso un'altra chiamata LVGL.

Come si fa?

Metodo 1: Utilizzare un Gateway Thread

Un "Gateway Thread" (o "Gateway Task" in alcuni sistemi operativi) è un thread (task) che il progettista del sistema designa per gestire *esclusivamente* una risorsa di sistema. Un esempio è la gestione di un chip remoto, come una EEPROM o un altro dispositivo, che deve sempre essere portato in uno stato coerente prima dell'avvio di qualcosa di nuovo. Un altro esempio è la gestione di più dispositivi su un bus I2C (o qualsiasi bus dati). In questo caso, il bus I2C è la "risorsa gestita in modo esclusivo" e avere un solo thread che lo gestisce garantisce che ogni azione avviata possa essere completata prima che venga avviata un'altra azione con essa.

Le strutture dati di LVGL sono una risorsa di sistema che richiede tale protezione.

Utilizzando questo metodo, la creazione, la modifica e l'eliminazione di tutti i widget e delle altre risorse LVGL (ovvero tutte le chiamate di funzione LVGL, escluse le *eccezioni* menzionate in precedenza) vengono chiamate da quel thread. Ciò significa che il thread è anche l'UNICO chiamante di `lv_timer_handler()`. (*Aggiungere LVGL al Proprio Progetto* per ulteriori informazioni).

Questo garantisce che le strutture dati di LVGL "appaiano" *atomiche* (tutti i thread che utilizzano questi dati li "vedono" in uno stato coerente) perché nessun altro thread "visualizza" tali strutture dati. Questo è imposto dalla disciplina del programmatore che garantisce che il *Gateway Thread* sia l'unico thread a effettuare chiamate LVGL (escluse le *exceptions* menzionate sopra).

Se i *dati atomici* relativi all'interfaccia utente vengono aggiornati in un altro thread (ad esempio da un altro task o in un interrupt), il thread che chiama le funzioni LVGL può leggere tali dati direttamente senza preoccuparsi che

siano in uno stato incoerente. (Per evitare un inutile sovraccarico della CPU, è possibile fornire un meccanismo [ad esempio, un flag generato dal thread di aggiornamento] in modo che l'interfaccia utente venga aggiornata solo quando ciò comporterà una modifica visibile all'utente finale).

Se i *dati non atomici* relativi all'interfaccia utente vengono aggiornati in un altro thread (ad esempio, da un altro task o durante un'interruzione), un modo alternativo (e sicuro) per trasmettere tali dati al thread che chiama le funzioni LVGL consiste nel passare una copia privata di tali dati a tale thread tramite una coda o un altro meccanismo del sistema operativo che protegga tali dati dalla visualizzazione in uno stato incoerente.

L'utilizzo di un *Gateway Thread* evita il sovraccarico di CPU (e di codice) derivante dall'utilizzo di un MUTEX per proteggere le strutture dati LVGL.

Metodo 2: Utilizzare un MUTEX

Un MUTEX sta per "MUTually EXclusive" (reciprocamente esclusivo) ed è una primitiva di sincronizzazione che protegge lo stato di una risorsa di sistema da modifiche o accessi simultanei da parte di più thread di esecuzione. In altre parole, fa sì che i dati così protetti "appaiano" atomici (tutti i thread che utilizzano questi dati li "vedono" in uno stato coerente). La maggior parte dei sistemi operativi fornisce MUTEX.

Il progettista del sistema assegna un singolo MUTEX per produrre una singola risorsa di sistema. Una volta assegnato, il MUTEX esegue tale protezione da parte dei programmatori:

1. acquisendo il MUTEX (ovvero bloccandolo) prima di accedere o modificare tale risorsa, e
2. rilasciando il MUTEX (ovvero sbloccandolo) al termine dell'accesso o della modifica.

Se un thread tenta di acquisire (lock) il MUTEX mentre un altro thread ne è "proprietario", quel thread attende che l'altro thread lo rilasci (unlock) prima di poter continuare l'esecuzione.

Per essere chiari: questo deve essere fatto *sia* dai thread che LEGGONO da quella risorsa, sia dai thread che MODIFICANO quella risorsa.

Se un MUTEX viene utilizzato per proteggere le strutture dati LVGL, ciò significa *ogni* chiamata di funzione LVGL (o gruppo di chiamate di funzione) deve essere preceduta da #1 e seguita da #2, incluse le chiamate a `lv_timer_handler()`.

> **Nota**
>
> Se il sistema operativo è integrato con LVGL (la macro `LV_USE_OS` ha un valore diverso da `LV_OS_NONE` in `lv_conf.h`) si possono usare `lv_lock()` e `lv_unlock()` per eseguire i punti #1 e #2.
>
> In questo caso, `lv_timer_handler()` chiama internamente `lv_lock()` e `lv_unlock()`, quindi non si devono includere le chiamate a `lv_timer_handler()` tra di esse.
>
> Se il sistema operativo NON è integrato con LVGL, queste chiamate o terminano immediatamente senza alcun effetto, oppure vengono ottimizzate dal linker.
>
> Per abilitare `lv_lock()` e `lv_unlock()`, imposta `LV_USE_OS` su un valore diverso da `LV_OS_NONE`.

Questo pseudocodice illustra il concetto di utilizzo di un MUTEX:

```
void lvgl_thread(void)
{
    while(1) {
        uint32_t time_till_next;
        time_till_next = lv_timer_handler(); /* lv_lock/lv_unlock is␣
↪called internally */
        if(time_till_next == LV_NO_TIMER_READY) time_till_next = LV_DEF_
↪REFR_PERIOD; /*try again soon because the other thread can make the␣
↪timer ready*/
        thread_sleep(time_till_next); /* sleep for a while */
    }
```

(continues on next page)

(continua dalla pagina precedente)

```
}

void other_thread(void)
{
    /* You must always hold (lock) the MUTEX while calling LVGL functions.␣
↪*/
    lv_lock();
    lv_obj_t *img = lv_image_create(lv_screen_active());
    lv_unlock();

    while(1) {
        lv_lock();
        /* Change to next image. */
        lv_image_set_src(img, next_image);
        lv_unlock();
        thread_sleep(2000);
    }
}
```

Gestione della Modalità Sleep

L'MCU può entrare in modalità sleep quando non riceve input dall'utente per un certo periodo di tempo. In questo caso, il `while(1)` principale potrebbe apparire così:

```
while(1) {
    /* Normal operation (no sleep) in < 1 sec inactivity */
    if(lv_display_get_inactive_time(NULL) < 1000) {
        lv_timer_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);
}
```

Si devono anche aggiungere le seguenti righe alla funzione di lettura del dispositivo di input per segnalare l'avvenuta riattivazione (pressione, tocco, clic, ecc.):

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

Oltre a *`lv_display_get_inactive_time()`* si può controllare *`lv_anim_count_running()`* per verificare se tutte le animazioni sono terminate.

4.1.7 Altre Piattaforme e Tool

Integrazione per vedere come utilizzare LVGL su diverse piattaforme. Lì si troveranno molte descrizioni specifiche per ogni piattaforma, ad esempio per ESP32, Arduino, NXP, RT-Thread, NuttX, ecc.

4.2 Binding

4.2.1 API di Output come Dati JSON

A partire dal 20 giugno 2024, LVGL viene fornito con uno script Python (`./scripts/gen_json/gen_json.py`) che legge i file di header in LVGL e restituisce un formato JSON più intuitivo per l'API. Questo viene fatto in modo che i binding che generano codice automaticamente possano raccogliere facilmente le informazioni necessarie senza dover reinventare la ruota. Il formato JSON è stato scelto perché esistono librerie per la lettura di dati JSON in quasi tutti i linguaggi di programmazione.

Requisiti

- Python >= 3.10
- Pycparser >= 2.22: Libreria Python per leggere l'output del preprocessore C
- PyMSVC >= 0.4.0: Libreria Python per l'utilizzo del compilatore MSVC
- Un compilatore C: gcc per Linux, clang per OSX e MSVC per Windows
- Doxygen: utilizzato per leggere i commenti di Doxygen (la documentazione dell'API) dai file di intestazione.

Utilizzo

Opzioni della Riga di Comando

- `--output-path`: directory di output per il file JSON. Se non viene specificato, l'output verrà visualizzato su stdout.
- `--lvgl-config`: path per lv_conf.h (incluso il nome del file). Se non è impostato, verrà generato un file di configurazione con le opzioni LVGL più comuni attivate.
- `--develop`: lascia i file generati nella cartella temporanea.

Esempi

Utilizzo normale:

```
python ./scripts/gen_json/gen_json.py --output-path=json/output/directory -
↪-lvgl-config=path/to/lv_conf.h
```

Per eseguire un sottoprocesso dall'interno di uno script di generazione e leggere l'output da stdout:

```
python ./scripts/gen_json/gen_json.py --lvgl-config=path/to/lv_conf.h
```

Dati di Output

Il contenuto del file di output è un oggetto JSON di grandi dimensioni (`{...}`) con le seguenti coppie chiave/valore (queste sono le chiavi):

```
{
    "enums"            : [...],
    "functions"        : [...],
    "function_pointers": [...],
    "structures"       : [...],
    "unions"           : [...],
    "variables"        : [...],
    "typedefs"         : [...],
    "forward_decls"    : [...],
    "macros"           : [...]
}
```

Come si vede, il valore di ciascuno di questi elementi è un array. Gli elementi di ogni array sono oggetti JSON, ciascuno con una struttura univoca per il tipo indicato dal nome dell'elemento padre (ad esempio "enums", "functions", ecc.).

A ciascun oggetto è stata aggiunta una coppia chiave/valore (chiave = "json_type") per consentire il passaggio di un oggetto a una funzione generica e far sì che ciascun oggetto conosca il proprio tipo tramite questo campo. I possibili valori di "json_type" sono:

- `"array"`: Il tipo dell'array viene utilizzato per identificare gli array.

  Campi:

  - `"dim"`: numero di elementi nell'array
  - `"quals"`: array di qualificatori, ad esempio, "const"
  - `"type"`: Questo potrebbe essere disponibile o meno.
  - `"name"`: nome del tipo di dati

- `"field"`: Questo tipo viene utilizzato per descrivere i campi in strutture e unioni. Viene utilizzato nell'array `"fields"` dei tipi `"struct"` `"union"` trattati di seguito.

  Campi:

  - `"name"`: nome del campo
  - `"type"`: data type
  - `"json_type"` carries object type (e.g. "enum", "function", etc.) identifying the top-level group it comes from
  - `"bitsize"`: Numero di bit per i campi bit, o `null` per i tipi di campo normali.
  - `"docstring"`: stringa contenente la documentazione estratta da Doxygen.

- `"arg"`: Descrive un argomento di funzione.

  Campi:

  - `"name"`: nome dell'argomento
  - `"type"`: data type
  - `"json_type"` carries object type (e.g. "enum", "function", etc.) identifying the top-level group it comes from.
  - `"docstring"`: stringa contenente la documentazione estratta da Doxygen.
  - `"quals"`: array di tutti i qualificatori presenti, ad esempio "const"

- `"forward_decl"`: Descrive una dichiarazione in avanti (forward). In LVGL ci sono strutture considerate private, ed è proprio ciò che descrivono.

  Campi:

  - `"name"`: nome della dichiarazione forward
  - `"type"`: data type
  - `"json_type"` carries object type (e.g. "enum", "function", etc.) identifying the top-level group it comes from.
  - `"docstring"`: stringa contenente la documentazione estratta da Doxygen.
  - `"quals"`: array di tutti i qualificatori presenti, ad esempio "const"

- `"function_pointer"`: Descrive un puntatore a funzione. Questi vengono utilizzati per la registrazione di funzioni di callback in LVGL.

  Campi:

  - `"name"`: nome del puntatore a funzione

  - `"type"`: tipo di ritorno della funzione
  - `"docstring"`: stringa contenente la documentazione estratta da Doxygen.
  - `"args"`: array degli oggetti `"arg"` descritti sopra
  - `"quals"`: array di tutti i qualificatori presenti, ad esempio "const"
- `"variable"`: Descrive una variabile globale.

  Campi:

  - `"name"`: nome della variabile
  - `"type"`: data type
  - `"json_type"` carries object type (e.g. "enum", "function", etc.) identifying the top-level group it comes from.
  - `"docstring"`: stringa contenente la documentazione estratta da Doxygen.
  - `"quals"`: array di tutti i qualificatori presenti, ad esempio "const"
  - `"storage"`: array di tutti gli specificatori di classe di archiviazione presenti (ad esempio "auto", "static", "extern", ecc.)
- `"special_type"`: Attualmente utilizzato solo per descrivere un argomento con ellissi di una funzione.

  Campi:

  - `"name"`: sempre "ellissi"
- `"primitive_type"`: Tipo di dati che non inizia con `"lv_"` e non termina con `"_t"`. Confrontare con `"lvgl_type"`. Include struct, union, tipi interi (ad esempio int, unsigned int), ecc.

  Campi:

  - `"name"`: nome del tipo primitivo
- `"enum"`: Enumerazioni C

  Campi:

  - `"name"`: Se l'enumerazione è il risultato di un `typedef`, questo campo riporta il nome del tipo definito. Esempio: `lv_align_t`. (Non sempre disponibile).
  - `"type"`: tipo di enumeratori (sempre "int")
  - `"docstring"`: stringa contenente la documentazione estratta da Doxygen.
  - `"members"`: array di oggetti `"enum_member"`
- `"enum_member"`: enumeratore (valore di enumerazione). Questo "json_type" si trova solo nell'array `"members"` di un oggetto `"enum"`

  Campi:

  - `"name"`: nome dell'enumeratore
  - `"type"`: Se l'enumerazione è il risultato di un `typedef`, questo campo contiene il nome del tipo definito. Esempio: `lv_align_t`.
  - `"docstring"`: stringa contenente la documentazione estratta da Doxygen.
  - `"value"`: valore dell'enumeratore
- `"lvgl_type"`: Tipo di dati definito in LVGL (inizia con `"lv_"` e termina con `"_t"`.

  Campi:

  - `"name"`: nome del tipo
  - `"quals"`: array di tutti i qualificatori presenti, ad esempio "const"

- `"struct"`: struct C

  Campi:

  - `"name"`: nome della struct (tipo di dati se definito da `typedef`)
  - `"type"`: un oggetto "primitive_type" object {"name": "struct", "json_type": "primitive_type"}. (Vedere la definizione sopra).
  - `"docstring"`: stringa contenente la documentazione estratta da Doxygen.
  - `"fields"`: array di oggetti `"field"` (Vedere la definizione sopra).
  - `"quals"`: array di tutti i qualificatori presenti, ad esempio "const"
- `"union"`: union C

  Campi:

  - `"name"`: nome della union (tipo di dati se definito da `typedef`)
  - `"type"`: un oggetto "primitive_type" {"name": "union", "json_type": "primitive_type"}. (Vedere la definizione sopra).
  - `"docstring"`: stringa contenente la documentazione estratta da Doxygen.
  - `"fields"`: array di elementi `"field"`.
  - `"quals"`: array di tutti i qualificatori presenti, ad esempio "const"
- `"macro"`: macro C. Le informazioni che si possono raccogliere sulle macro sono limitate e nella maggior parte dei casi un binding richiederà che queste vengano aggiunte staticamente a un binding. Serve più per raccogliere le docstring che altro.

  Campi:

  - `"name"`: nome della macro
  - `"docstring"`: stringa contenente la documentazione estratta da Doxygen.
- `"ret_type"`: tipo di ritorno da una funzione. Questo verrà visualizzato solo nell'elemento `"type"` di un tipo `"function"`.

  Campi:

  - `"type"`: data type
  - `"json_type"` carries object type (e.g. "enum", "function", etc.) identifying the top-level group it comes from.
  - `"docstring"`: stringa contenente la documentazione estratta da Doxygen.
- `"function"`: funzione C

  Campi:

  - `"name"`: nome della funzione
  - `"type"`: Un oggetto "ret_type". (Vedere la definizione sopra).
  - `"docstring"`: stringa contenente la documentazione estratta da Doxygen.
  - `"args"`: array of `"arg"` tipi json. (Vedere la definizione sopra).
- `"stdlib_type"`: tipo intero C (int, unsigned int, float, ecc.)

  Campi:

  - `"name"`: nome del tipo
  - `"quals"`: array di tutti i qualificatori presenti, ad esempio "const"

- `"unknown_type"`: Questo non dovrebbe essere visibile. In tal caso, è necessario apportare una modifica allo script. Aprire una segnalazione per farci sapere se si vede questo tipo.

  Campi:

  - `"name"`: nome del tipo
  - `"quals"`: array di tutti i qualificatori presenti, ad esempio "const"
- `"pointer"`: puntatore C

  Campi:

  - `"type"`: pointer type
  - `"json_type"` carries object type (e.g. "enum", "function", etc.) identifying the top-level group it comes from.
  - `"quals"`: array di tutti i qualificatori presenti, ad esempio "const"
- `"typedef"`: definizione del tipo C

  Campi:

  - `"name"`: nome del tipo (ad esempio `lv_part_t`)
  - `"type"`: un oggetto "primitive_type" {"name": "uint32_t", "json_type": "stdlib_type"}. (Vedere la definizione sopra).
  - `"json_type"` carries object type (e.g. "enum", "function", etc.) identifying the top-level group it comes from.
  - `"docstring"`: stringa contenente la documentazione estratta da Doxygen.
  - `"quals"`: array di tutti i qualificatori presenti, ad esempio "const"

Ecco un esempio abbreviato di come appare l'output.

```
{
    "enums":[
        {
            "name":"_lv_result_t",
            "type":{
                "name":"int",
                "json_type":"primitive_type"
            },
            "json_type":"enum",
            "docstring":"LVGL error codes. ",
            "members":[
                {
                    "name":"LV_RESULT_INVALID",
                    "type":{
                        "name":"_lv_result_t",
                        "json_type":"lvgl_type"
                    },
                    "json_type":"enum_member",
                    "docstring":"",
                    "value":"0x0"
                },
                {
                    "name":"LV_RESULT_OK",
                    "type":{
                        "name":"_lv_result_t",
                        "json_type":"lvgl_type"
                    },
```

(continues on next page)

(continua dalla pagina precedente)


```
                    "json_type":"enum_member",
                    "docstring":"",
                    "value":"0x1"
                }
            ]
        }
    ],
    "functions":[
        {
            "name":"lv_version_info",
            "type":{
                "type":{
                    "type":{
                        "name":"char",
                        "json_type":"primitive_type",
                        "quals":[
                            "const"
                        ]
                    },
                    "json_type":"pointer",
                    "quals":[]
                },
                "json_type":"ret_type",
                "docstring":""
            },
            "json_type":"function",
            "docstring":"",
            "args":[
                {
                    "name":null,
                    "type":{
                        "name":"void",
                        "json_type":"primitive_type",
                        "quals":[]
                    },
                    "json_type":"arg",
                    "docstring":"",
                    "quals":[]
                }
            ]
        }
    ],
    "function_pointers":[
        {
            "name":"lv_tlsf_walker",
            "type":{
                "type":{
                    "name":"void",
                    "json_type":"primitive_type",
                    "quals":[]
                },
                "json_type":"ret_type",
                "docstring":""
            },
            "json_type":"function_pointer",
            "docstring":"",
```


(continues on next page)

(continua dalla pagina precedente)

```
            "args":[
                {
                    "name":"ptr",
                    "type":{
                        "type":{
                            "name":"void",
                            "json_type":"primitive_type",
                            "quals":[]
                        },
                        "json_type":"pointer",
                        "quals":[]
                    },
                    "json_type":"arg",
                    "docstring":""
                },
                {
                    "name":"size",
                    "type":{
                        "name":"size_t",
                        "json_type":"stdlib_type",
                        "quals":[]
                    },
                    "json_type":"arg",
                    "docstring":""
                },
                {
                    "name":"used",
                    "type":{
                        "name":"int",
                        "json_type":"primitive_type",
                        "quals":[]
                    },
                    "json_type":"arg",
                    "docstring":""
                },
                {
                    "name":"user",
                    "type":{
                        "type":{
                            "name":"void",
                            "json_type":"primitive_type",
                            "quals":[]
                        },
                        "json_type":"pointer",
                        "quals":[]
                    },
                    "json_type":"arg",
                    "docstring":""
                }
            ],
            "quals":[]
        }
    ],
    "structures":[
        {
            "name":"_lv_grad_cache_t",
```

(continues on next page)

(continua dalla pagina precedente)

```
            "type":{
                "name":"struct",
                "json_type":"primitive_type"
            },
            "json_type":"struct",
            "docstring":null,
            "fields":[
                {
                    "name":"color_map",
                    "type":{
                        "type":{
                            "name":"lv_color_t",
                            "json_type":"lvgl_type",
                            "quals":[]
                        },
                        "json_type":"pointer",
                        "quals":[]
                    },
                    "json_type":"field",
                    "bitsize":null,
                    "docstring":""
                },
                {
                    "name":"opa_map",
                    "type":{
                        "type":{
                            "name":"lv_opa_t",
                            "json_type":"lvgl_type",
                            "quals":[]
                        },
                        "json_type":"pointer",
                        "quals":[]
                    },
                    "json_type":"field",
                    "bitsize":null,
                    "docstring":""
                },
                {
                    "name":"size",
                    "type":{
                        "name":"uint32_t",
                        "json_type":"stdlib_type",
                        "quals":[]
                    },
                    "json_type":"field",
                    "bitsize":null,
                    "docstring":""
                }
            ]
        }
    ],
    "unions":[],
    "variables":[
        {
            "name":"lv_global",
            "type":{
```

(continues on next page)

(continua dalla pagina precedente)


```
                "name":"lv_global_t",
                "json_type":"lvgl_type",
                "quals":[]
            },
            "json_type":"variable",
            "docstring":"",
            "quals":[],
            "storage":[
                "extern"
            ]
        }
    ],
    "typedefs":[
        {
            "name":"lv_pool_t",
            "type":{
                "type":{
                    "name":"void",
                    "json_type":"primitive_type",
                    "quals":[]
                },
                "json_type":"pointer"
            },
            "json_type":"typedef",
            "docstring":"",
            "quals":[]
        }
    ],
    "forward_decls":[
        {
            "name":"lv_fragment_managed_states_t",
            "type":{
                "name":"struct",
                "json_type":"primitive_type"
            },
            "json_type":"forward_decl",
            "docstring":"",
            "quals":[]
        }
    ],
    "macros":[
        {
            "name":"ZERO_MEM_SENTINEL",
            "json_type":"macro",
            "docstring":""
        }
    ]
}
```


4.2.2 Cpp

In progress: https://github.com/lvgl/lv_binding_cpp

4.2.3 JavaScript

Con lv_binding_js si può usare LVGL dall'interno di JavaScript.

Sfrutta il concetto di DOM virtuale di React per manipolare i componenti dell'interfaccia utente LVGL, offrendo agli utenti un'esperienza familiare simile a React.

Codice

Codice in Esecuzione su Dispositivo Reale

Funzionalità

- Supporta tutti i componenti LVGL nativi
- Supporta pienamente gli stili flex e grid LVGL
- Supporta la maggior parte degli stili LVGL, basta scrivere come CSS HTML5
- Supporta il caricamento dinamico delle immagini
- Supporta pienamente le animazioni LVGL

Demo

Vedere la cartella demo

Building

Di seguito sono riportate alcune note per sviluppatori su come compilare lvgljs sulla piattaforma nativa. Non si tratta di guide complete, ma includono note sulle librerie necessarie, i flag di compilazione, ecc.

lvgljs

- Note di build per dispositivi Linux embedded
- Note di build per SDL Simulator (Linux e macOS)

Bundle JS

- Note di build del bundle JS

Components

- View
- Image
- Button
- Text
- Input
- Textarea
- Switch
- Checkbox
- Dropdownlist
- ProgressBar
- Line
- Roller
- Keyboard

- Calendar
- Chart

Font

- Builtin-Symbol

Animation

- Animation

Style

- position-size-layout
- boxing-model
- color
- flex
- grid
- font
- opacity
- display
- background
- scroll
- shadow
- recolor
- line
- transition
- transform

JSAPI

- network
- filesystem
- dimension

Thanks

lvgljs depends on following excellent work:

- lvgl: Create beautiful UIs for any MCU, MPU and display type
- QuickJS: JavaScript engine
- libuv: platform abstraction layer
- curl: HTTP client
- txiki.js: Tiny JavaScript runtime

4.2.4 MicroPython

Cos'è MicroPython?

MicroPython è Python per microcontrollori. Utilizzando MicroPython, è possibile scrivere codice Python3 ed eseguirlo anche su un'architettura bare metal con risorse limitate. Una delle sue potenti funzionalità è la possibilità di modificare il comportamento di un dispositivo modificando il codice Python su una memoria rimovibile (o interna), senza dover modificare il firmware del dispositivo.

Punti di Forza di MicroPython

Compatto
Si adatta e funziona in soli 256 kB di spazio di codice e 16 kB di RAM. Non è necessario alcun sistema operativo, anche se è possibile eseguirlo anche con un sistema operativo, volendo.

Compatibile
Cerca di essere il più compatibile possibile con Python standard (noto come CPython).

Versatile
Supporta molte architetture (x86, x86-64, ARM, ARM Thumb, Xtensa).

Interattivo
Non è necessario il ciclo di compilazione-flash-boot. Con REPL (prompt interattivo) è possibile digitare comandi ed eseguirli immediatamente, eseguire script, ecc.

Popolare
Sono supportate numerose piattaforme. La base utenti è in continua crescita. Fork degne di nota:

- MicroPython
- CircuitPython
- MicroPython_ESP32_psRAM_LoBo

Orientato agli Embedded
Fornito con moduli specifici per sistemi embedded, come il modulo machine per l'accesso all'hardware di basso livello (pin I/O, ADC, UART, SPI, I2C, RTC, timer ecc.)

Perché MicroPython + LVGL?

MicroPython non dispone di una buona libreria GUI nativa di alto livello. LVGL è una libreria GUI di alto livello Basata su Componenti Object-Oriented che è una candidata naturale per essere mappata in un linguaggio di livello superiore, come Python. LVGL è implementato in C e le sue API sono in C.

Ecco alcuni vantaggi dell'utilizzo di LVGL in MicroPython:

- Sviluppo di GUI in Python, un linguaggio di alto livello molto diffuso. Utilizzo di paradigmi come la Programmazione Object-Oriented.
- Di solito, lo sviluppo di GUI richiede più iterazioni per ottenere risultati ottimali. In C, ogni iterazione consiste in ``**Modifica del codice**`` > ``**Build**`` > ``**Flash**`` > ``**Run**``. In MicroPython è semplicemente ``**Modifica del codice**`` > ``**Run**`` ! È anche possibile eseguire comandi in modo interattivo utilizzando REPL (il prompt interattivo)

MicroPython + LVGL potrebbe essere utilizzato per:

- Prototipazione rapida di GUI
- Ridurre il ciclo di modifica e messa a punto di una GUI
- Modellare una GUI in modo più astratto definendo widget compositi riutilizzabili, sfruttando le funzionalità del linguaggio Python come ereditarietà, chiusure, comprensione di liste, generatori, gestione delle eccezioni, interi a precisione arbitraria e altre.

- Rendere LVGL accessibile a un pubblico più vasto. Non è necessario conoscere il C per creare una buona GUI su un sistema embedded. Questo si integra perfettamente con la visione CircuitPython. CircuitPython è stato progettato pensando alla formazione, per facilitare l'approccio allo sviluppo embedded da parte di programmatori nuovi o inesperti.
- Creare strumenti per lavorare con LVGL a un livello superiore (ad esempio, un designer drag-and-drop).

Che aspetto ha?

È molto simile all'API C, ma orientata agli oggetti per i componenti LVGL.

Immergiamoci subito in un esempio!

Un semplice esempio

```
# Initialize
import display_driver
import lvgl as lv

# Create a button with a label
scr = lv.obj()
btn = lv.button(scr)
btn.align(lv.ALIGN.CENTER, 0, 0)
label = lv.label(btn)
label.set_text('Hello World!')
lv.screen_load(scr)
```

Come si Usa?

Simulatore Online

Per sperimentare con LVGL + MicroPython senza scaricare nulla, si può usare il simulatore online! È un simulatore LVGL + MicroPython completamente funzionante che funziona interamente nel browser e permette di modificare uno script Python ed eseguirlo.

Cliccare qui per sperimentare sul simulatore online

Molti *esempi LVGL* sono disponibili anche per MicroPython. Basta cliccare sul link!

Simulatore per PC

MicroPython è disponibile su numerose piattaforme. Una versione degna di nota è quella per "Unix", che consente di compilare ed eseguire MicroPython (+LVGL) su una macchina Linux. (Su una macchina Windows potrebbe essere necessario Virtual Box, WSL, MinGW, Cygwin, ecc.)

Cliccare qui per saperne di più sulla compilazione e l'esecuzione della versione Unix.

Piattaforme Embedded

In definitiva, l'obiettivo è eseguire il tutto su una piattaforma embedded. Sia MicroPython che LVGL possono essere utilizzati su numerose architetture embedded. lv_micropython è un fork di MicroPython+LVGL e attualmente supporta Linux, ESP32, STM32 e RP2. Può essere portato su qualsiasi altra piattaforma supportata da MicroPython.

- Sono necessari anche driver di input e display. È possibile utilizzare uno dei driver esistenti forniti con lv_micropython oppure creare driver di input/display personalizzati per l'hardware specifico.
- I driver possono essere implementati in C come modulo MicroPython o in Python puro.

lv_micropython contiene già i seguenti driver:

- Driver per display:
  - SDL su Linux
  - X11 su Linux
  - Specifici per ESP32:
    - ILI9341
    - ILI9488
    - GC9A01
    - ST7789
    - ST7735
  - Generici (Python puro):
    - ILI9341
    - ST7789
    - ST7735
- Driver di input:
  - SDL
  - X11
  - XPT2046
  - FT6X36
  - ESP32 ADC con touch resistivo

Dove si possono trovare maggiori informazioni?

- `lv_micropython` README
- `lv_binding_micropython` README
- Il forum LVGL micropython (Sentitevi liberi di chiedere qualsiasi cosa!)
- Su MicroPython: docs e forum
- Post del Blog, un po' obsoleto.

Il Binding MicroPython è generato automaticamente!

- LVGL è un sottomodulo git all'interno di lv_micropython (LVGL è un sottomodulo git di lv_binding_micropython che a sua volta è un sottomodulo di lv_micropython).
- Durante la compilazione di lv_micropython, l'API C pubblica di LVGL viene analizzata e l'API MicroPython viene generata automaticamente. Ciò significa che lv_micropython fornisce l'API LVGL per **qualsiasi** versione di LVGL e generalmente non richiede modifiche al codice man mano che LVGL si evolve.

Convenzioni di codifica dell'API C di LVGL

Per un riepilogo delle convenzioni di codifica da seguire, consultare *Stile di Codifica*.

Gestione della Memoria

- Quando LVGL viene eseguito in MicroPython, tutte le allocazioni di memoria dinamica (`lv_malloc()`) vengono gestite dal gestore di memoria di MicroPython, che è garbage-collected (GC).
- Per evitare che il GC raccolga la memoria prematuramente, tutta la RAM allocata dinamicamente deve essere raggiungibile dal GC.

- Il GC riconosce la maggior parte delle allocazioni, ad eccezione dei puntatori al Data Segment:
  - Puntatori che sono variabili globali
  - Puntatori che sono variabili globali statiche
  - Puntatori che sono variabili locali statiche

Tali puntatori devono essere definiti in modo specifico per renderli raggiungibili dal GC.

Identificare Il Problema

Un problema si verifica quando un puntatore di memoria allocata (valore di ritorno di `lv_malloc()`) viene memorizzato solo in una variabile puntatore **global**, **static global** o **static local** e non come parte di una `struct` precedentemente allocata o di un'altra variabile.

Risoluzione del Problema

- Sostituire la variabile locale globale/statica con `(LV_GLOBAL_DEFAULT()->_var)`
- Includere `lv_global.h` nei file che usano `LV_GLOBAL_DEFAULT`
- Aggiungere `_var` a `lv_global_t` su `lv_global.h`

Esempio

Ulteriori Informazioni sulla Gestione della Memoria

- Nel README
- Nel Blog

Callback

In C, un callback è semplicemente un puntatore a funzione. In MicroPython, invece, dobbiamo registrare un *oggetto richiamabile MicroPython* per ogni callback. Pertanto, nel binding MicroPython dobbiamo registrare sia un puntatore a funzione che un oggetto MicroPython per ogni callback.

Abbiamo quindi definito una **convenzione di callback** per l'API C LVGL che prevede che gli header LVGL siano definiti in un certo modo. Le callback dichiarate secondo questa convenzione consentono al binding di registrare un oggetto MicroPython accanto al puntatore a funzione quando si registra una callback e di accedere a tale oggetto quando la callback viene chiamata.

- L'idea di base è che abbiamo un campo `void * user_data` che viene utilizzato automaticamente dal binding MicroPython per salvare l'*oggetto richiamabile MicroPython* per una callback. Questo campo deve essere specificato durante la registrazione del puntatore alla funzione e fornito alla funzione di callback stessa.
- Sebbene chiamato "user_data", non è previsto che l'utente legga/scriva tale campo. Invece, il codice "glue" di MicroPython usa `user_data` per tenere traccia automaticamente dell'oggetto richiamabile di MicroPython. Il codice "glue" lo aggiorna quando la callback viene registrata e lo utilizza quando la callback viene chiamata per invocare una chiamata all'oggetto richiamabile originale.

Esistono alcune opzioni per definire un callback nell'API C LVGL:

- Opzione 1: `user_data` in una struct
  - Esiste una struttura che contiene un campo chiamato `void * user_data`
    - Un puntatore a tale struttura viene fornito come **primo** argomento di una funzione di registrazione della callback.
    - Un puntatore a tale struttura viene fornito come **primo** argomento della callback stessa.
- Opzione 2: `user_data` come argomento della funzione
  - Un parametro chiamato `void * user_data` viene fornito alla funzione di registrazione come **ultimo** argomento

    * La callback stessa riceve `void *` come **ultimo** argomento

- Opzione 3: sia la callback che `user_data` sono campi di struct
  - L'API espone una struct con sia un membro puntatore a funzione che un membro `user_data`
    * Il membro puntatore a funzione riceve la stessa struct come suo **primo** argomento

In pratica è anche possibile combinare queste opzioni, ad esempio fornire un puntatore a struct quando si registra una callback (opzione 1) e fornire l'argomento `user_data` quando si chiama la callback (opzione 2), **purché lo stesso** `user_data` **registrato venga passato alla callback quando viene chiamata**.

Esempi

- `lv_anim_t` contiene il campo `user_data`. *`lv_anim_set_path_cb()`* registra la callback `path_cb`. Sia `lv_anim_set_path_cb` che *`lv_anim_path_cb_t`* ricevono `lv_anim_t` come primo argomento
- Il campo `path_cb` può anche essere assegnato direttamente nel codice Python perché è un membro di `lv_anim_t` che contiene il campo `user_data`, e *`lv_anim_path_cb_t`* riceve `lv_anim_t` come primo argomento.
- *`lv_imgfont_create()`* registra `path_cb` e riceve `user_data` come ultimo argomento. Anche la callback *`lv_imgfont_get_path_cb_t`* riceve `user_data` come ultimo argomento.

Ulteriori Approfondimenti sulle Callback

- Nel Blog e nel README
- [v6.0] Callback conventions #1036
- Varie discussioni: qui, qui e qui

4.2.5 PikaScript

Cos'è PikaScript?

PikaScript è un interprete Python progettato specificamente per microcontrollori e supporta un sottoinsieme della comune sintassi Python3.

È più leggero di MicroPython, richiedendo solo 32k di spazio codice e 4k di RAM, il che significa che può essere eseguito su stm32f103c8 (blue-pill) o persino stm32g030c8. D'altra parte, è possibile lasciare spazio prezioso per più materiale o aree di buffer più grandi.

È più semplice, pronto all'uso, funziona senza alcun porting o configurazione, non dipende dal sistema operativo o dal file system, offre un buon supporto per gli IDE più diffusi per piattaforme Windows come Keil, IAR, RT-Thread-Studio e, naturalmente, supporta le piattaforme di sviluppo Linux gcc.

È più intelligente, con un meccanismo esclusivo per i moduli C che consente di generare automaticamente i binding semplicemente scrivendo l'API per il modulo C in Python, senza dover affrontare il problema di dover scrivere manualmente macro o tabelle globali. D'altra parte, tutti i moduli C dispongono di suggerimenti intelligenti sofisticati, che suggeriscono persino i tipi degli argomenti.

Perché PikaScript + LVGL?

- PikaScript ora supporta le funzionalità principali di LVGL8 e queste API sono pienamente compatibili con MicroPython. Ciò significa che è possibile continuare a utilizzare il codice già scritto in MicroPython, ma occupando meno spazio di codice e RAM.
- Utilizzare suggerimenti dettagliati sul codice fino al tipo di parametro per una migliore esperienza di programmazione.
- Utilizzare un IDE più pratico, come i progetti di simulazione basati su Visual Studio.

Come si Presenta?

Ecco alcuni esempi di utilizzo di LVGL che PikaScript può già eseguire. Provengono principalmente dagli esempi della documentazione di LVGL.

LV_ARC

```
import pika_lvgl as lv
import PikaStdLib
mem = PikaStdLib.MemChecker()
# Create an Arc
arc = lv.arc(lv.screen_active())
arc.set_end_angle(200)
arc.set_size(150, 150)
arc.center()
print('mem used max: %0.2f kB' % (mem.getMax()))
print('mem used now: %0.2f kB' % (mem.getNow()))
```

LV_BAR

```
import pika_lvgl as lv
import PikaStdLib
mem = PikaStdLib.MemChecker()
bar1 = lv.bar(lv.screen_active())
bar1.set_size(200, 20)
bar1.center()
bar1.set_value(70, lv.ANIM.OFF)
print('mem used max: %0.2f kB' % (mem.getMax()))
print('mem used now: %0.2f kB' % (mem.getNow()))
```

LV_BTN

```
import pika_lvgl as lv
import PikaStdLib
mem = PikaStdLib.MemChecker()


def event_cb_1(evt):
    print('in evt1')
    print('mem used now: %0.2f kB' % (mem.getNow()))


def event_cb_2(evt):
    print('in evt2')
    print('mem used now: %0.2f kB' % (mem.getNow()))


btn1 = lv.btn(lv.screen_active())
btn1.align(lv.ALIGN.TOP_MID, 0, 10)
btn2 = lv.btn(lv.screen_active())
btn2.align(lv.ALIGN.TOP_MID, 0, 50)
btn1.add_event_cb(event_cb_1, lv.EVENT.CLICKED, 0)
btn2.add_event_cb(event_cb_2, lv.EVENT.CLICKED, 0)
print('mem used max: %0.2f kB' % (mem.getMax()))
print('mem used now: %0.2f kB' % (mem.getNow()))
```

LV_CHECKBOX

```
import pika_lvgl as lv
import PikaStdLib
mem = PikaStdLib.MemChecker()
cb = lv.checkbox(lv.screen_active())
cb.set_text("Apple")
cb.align(lv.ALIGN.TOP_LEFT, 0 ,0)
cb = lv.checkbox(lv.screen_active())
cb.set_text("Banana")
cb.add_state(lv.STATE.CHECKED)
cb.align(lv.ALIGN.TOP_LEFT, 0 ,30)
cb = lv.checkbox(lv.screen_active())
cb.set_text("Lemon")
cb.add_state(lv.STATE.DISABLED)
cb.align(lv.ALIGN.TOP_LEFT, 0 ,60)
cb = lv.checkbox(lv.screen_active())
cb.add_state(lv.STATE.CHECKED | lv.STATE.DISABLED)
cb.set_text("Melon")
cb.align(lv.ALIGN.TOP_LEFT, 0 ,90)
print('mem used max: %0.2f kB' % (mem.getMax()))
print('mem used now: %0.2f kB' % (mem.getNow()))
```

Come Funziona?

PikaScript dispone di un esclusivo strumento di binding intelligente per i moduli C.

Basta scrivere l'interfaccia Python in pika_lvgl.pyi (.pyi è un file di interfaccia Python).

```
# pika_lvgl.pyi
class arc(lv_obj):
    def set_end_angle(self, angle: int): ...
    def set_bg_angles(self, start: int, end: int): ...
    def set_angles(self, start: int, end: int): ...
```

Il precompilatore di PikaScript può poi associare automaticamente le seguenti funzioni C, semplicemente nominando le funzioni nel formato modulo_classe_metodo, senza alcun lavoro aggiuntivo, e tutto il binding e la registrazione vengono eseguiti automaticamente.

```
/* pika_lvgl_arc.c */
void pika_lvgl_arc_set_end_angle(PikaObj* self, int angle) {
    lv_obj_t* lv_obj = obj_getPtr(self, "lv_obj");
    lv_arc_set_end_angle(lv_obj, angle);
}
void pika_lvgl_arc_set_bg_angles(PikaObj *self, int start, int end){
    lv_obj_t* lv_obj = obj_getPtr(self, "lv_obj");
    lv_arc_set_bg_angles(lv_obj, start, end);
}
void pika_lvgl_arc_set_angles(PikaObj *self, int start, int end){
    lv_obj_t* lv_obj = obj_getPtr(self, "lv_obj");
    lv_arc_set_angles(lv_obj, start, end);
}
```

Per utilizzare il modulo, basta `import pika_lvgl` e il precompilatore analizzerà automaticamente main.py e associarà il modulo `pika_lvgl`.

```
$ ./rust-msc-latest-win10.exe
(pikascript) packages installed:
    pikascript-core==v1.10.0
    PikaStdLib==v1.10.0
    PikaStdDevice==v1.10.0
(pikascript) pika compiler:
  scanning main.py...
    binding pika_lvgl.pyi...
```

Il precompilatore è scritto in Rust, funziona su Windows e Linux ed è completamente open source.

Oltre ad associare i moduli C, il precompilatore compila gli script Python in bytecode nel PC, riducendo le dimensioni dello script e aumentandone la velocità.

Come si Usa?

Il repository di simulazione per Visual Studio è disponibile all'indirizzo https://github.com/pikasTech/lv_pikascript .

4.3 Sistemi di Build

4.3.1 make

LVGL può essere facilmente integrato in qualsiasi progetto Makefile aggiungendo `include lvgl/lvgl.mk` al `Makefile` principale.

4.3.2 CMake

Panoramica

CMake è un generatore di sistemi di build multipiattaforma. Viene utilizzato per integrare facilmente un progetto/libreria in un altro progetto. Offre inoltre la possibilità di configurare la build con diverse opzioni, di abilitare o disabilitare componenti o di integrare l'esecuzione di script personalizzati durante la fase di configurazione/build.

LVGL include CMake in modo nativo, il che significa che è possibile utilizzarlo per configurare e compilare LVGL direttamente o integrarlo in una build CMake di livello superiore.

Questo progetto utilizza i preset CMake per garantire una build semplice. Maggiori informazioni su Cmake Presets qui: https://cmake.org/cmake/help/latest/manual/cmake-presets.7.html

Prerequisiti

È necessario installare

- CMake con GNU make o Ninja (per le build Linux). Aggiungere ninja/make al PATH!
- I prerequisiti sono elencati in `scripts/install-prerequisites.sh/bat`
- Un interprete Python3 per utilizzare KConfig.

Come compilare questo progetto con cmake

Compilazione da riga di comando

Il modo più semplice per compilare LVGL con cmake è usare le chiamate da riga di comando:

```
# Method 1
cd <lvgl_repo>
mkdir build
cd build
cmake ..            # Configure phase
cmake --build .     # Build phase

# Method 2
cd <lvgl_repo>
cmake -B build      # Configure phase
cmake --build build # build phase
```

Compilazione con i preset di CMake

Un altro modo per compilare questo progetto è utilizzare il file CMakePresets.json fornito o passare le opzioni tramite la riga di comando. Il file CMakePresets.json descrive alcune configurazioni di CMake e la fase di compilazione. È un modo per utilizzare rapidamente un set di opzioni predefinite di CMake.

Per ora, sono disponibili i seguenti preset di configurazione:

- `windows-base`: Una configurazione Windows, che utilizza VS MSVC. Usa `lv_conf.h` come sistema di configurazione.
- `windows-kconfig`: Una configurazione Windows, che utilizza VS MSVC. Utilizza Kconfig come sistema di configurazione.
- `linux-base`: Una configurazione Linux, che utilizza Ninja e GCC. Usa `lv_conf.h` come sistema di configurazione.
- `linux-kconfig`: Una configurazione Linux, che utilizza Ninja e GCC. Utilizza Kconfig come sistema di configurazione.

E questi sono i preset della build:

- `windows-base_dbg`: Build di Debug per Windows.
- `windows-base_rel`: Build di Release per Windows.
- `linux-base_dbg`: Build di Debug per Linux.
- `linux-base_rel`: Build di Release per Linux.

Ecco come compilare utilizzando i preset:

```
cmake --preset windows-base
cmake --build --preset windows-base_dbg
ctest --preset windows-base_dbg
```

Compilazione con IDE

Il metodo consigliato per utilizzare CMakePresets è un IDE compatibile con CMakePresets come

- VS 2022
- VS Code
- CLion

Basta caricare questo progetto nel proprio IDE e selezionare il preset desiderato.

Compilazione con la GUI di CMake

Aprire questo progetto con l'interfaccia grafica di CMake e selezionare il preset desiderato. Premendo il pulsante *generate*, CMake creerà file di soluzione (per VS) o file Ninja (per la build Ninja per Linux).

Sono disponibili i seguenti target.

- lvgl (la libreria vera e propria, obbligatoria)
- lvgl_thorvg (un'estensione per la grafica vettoriale, facoltativa)
- lvgl_examples (esempi di utilizzo, facoltativi)
- lvgl_demos (alcune demo, facoltative)

Tutti i target opzionali possono essere disabilitati impostando le variabili di cache appropriate. Se si utilizza cmake per installare lvgl, verranno create 3 cartelle.

- include/lvgl (contiene tutti gli header pubblici)
- bin (contiene tutti i binari (*.dll))
- lib (contiene tutti i file sorgente precompilati (*.lib))

Integrare LVGL nel progetto usando cmake

Il sistema cmake di LVGL è progettato per essere integrato in progetti di livello superiore. Per farlo, è sufficiente aggiungerlo al file `CMakeLists.txt` del progetto.

Questo frammento aggiunge LVGL e necessita di un file `lv_conf.h` presente accanto alla cartella lvgl:

```
add_subdirectory(lvgl)
```

Questo frammento imposta LVGL e gli indica quale file `lv_conf.h` utilizzare:

```
set(LV_BUILD_CONF_PATH path/to/my_lv_conf.h)
add_subdirectory(lvgl)
```

Questo frammento imposta LVGL e punta alla cartella in cui si trova `lv_conf.h`:

```
set(LV_BUILD_CONF_DIR path/to/directory)
add_subdirectory(lvgl)
```

Questo frammento aggiunge LVGL e specifica di utilizzare Kconfig come sistema di configurazione:

```
set(LV_BUILD_USE_KCONFIG ON)
add_subdirectory(lvgl)
```

Questo frammento aggiunge LVGL e specifica di utilizzare Kconfig come sistema di configurazione e di utilizzare un defconfig specifico:

```
set(LV_BUILD_USE_KCONFIG ON)
set(LV_BUILD_DEFCONFIG_PATH path/to/my_defconfig)
add_subdirectory(lvgl)
```

Per abilitare le demo e gli esempi, impostare queste opzioni:

```
set(CONFIG_LV_BUILD_EXAMPLES ON)
set(CONFIG_LV_BUILD_DEMOS ON)
add_subdirectory(lvgl)
```

Di seguito è riportato un elenco delle opzioni/variabili disponibili

Variabile/Opzione	Tipo	Descrizione
LV_BUILD_CONF_PATH	PATH	Consente di impostare un percorso personalizzato per `lv_conf.h`
LV_BUILD_CONF_DIR	PATH	Consente di impostare una directory contenente `lv_conf.h`
LV_BUILD_USE_KCONFIG	BOOLEAN	Quando impostato, KConfig viene utilizzato come sorgente della configurazione. Questa opzione è disabilitata per default.
LV_BUILD_DEFCONFIG_PATH	PATH	Specifica di utilizzare un file .defconfig al posto del file .config corrente in una configurazione di Kconfig.
LV_BUILD_LVGL_H_SYSTEM_INCLUDE	BOOLEAN	Abilitare se LVGL verrà installato sul sistema o se il sistema di build utilizza una sysroot. L'attivazione di questa opzione implica che le risorse generate dallo script di generazione dell'immagine includeranno `lvgl.h` come inclusione di sistema. Ad esempio: `#include <lvgl.h>`. Questa opzione è disabilitata per default.
LV_BUILD_LVGL_H_SIMPLE_INCLUDE	BOOLEAN	Se abilitata, le risorse includeranno `lvgl.h` come semplice inclusione; questa opzione è abilitata di default.
LV_BUILD_SET_CONFIG_OPTS	BOOLEAN	Se abilitata, questa opzione esegue uno script che elabora la configurazione `lv_conf.h`/Kconfig utilizzando `pcpp` per generare le variabili CMake `CONFIG_LV_*` e `CONFIG_LV_BUILD_*` corrispondenti in base al contenuto di `lv_conf_internal.h`. Questo richiede python3 con `venv` e `pip` o l'accesso a un `pcpp` funzionante. Se si utilizza KConfig, questa opzione viene abilitata automaticamente.
CONFIG_LV_BUILD_DEMOS	BOOLEAN	Se abilitato, compila le demo
CONFIG_LV_BUILD_EXAMPLES	BOOLEAN	Se abilitato, compila gli esempi
CONFIG_LV_USE_THORVG_INTERNAL	BOOLEAN	Se abilitato, viene compilata la versione LVGL in-tree di ThorVG
CONFIG_LV_USE_PRIVATE_API	BOOLEAN	Se abilitato, vengono installati sul sistema gli header privati `*_private.h`

Nota

Se `LV_BUILD_SET_CONFIG_OPTS` o `LV_BUILD_USE_KCONFIG` sono abilitati, le opzioni/variabili che iniziano con il prefisso `CONFIG_*` vengono automaticamente impostate sui valori presenti in `lv_conf.h`

4.4 Fornitori di chip

4.4.1 Alif

Alif Semiconductor è un fornitore di chip di nuova generazione che produce chip disponibili in una varietà di configurazioni. I loro chip sono progettati per eccellere sotto diversi aspetti contemporaneamente. Offrono combinazioni di prestazioni, basso consumo energetico, sicurezza e funzionalità speciali come l'intelligenza artificiale. Molti dei loro chip hanno due core asimmetrici. Un core è in genere ad alte prestazioni, mentre l'altro ad alta efficienza. Alif offre sia microcontrollori che microprocessori.

LVGL su schede Alif

Questa è una guida per iniziare a usare LVGL su una scheda Alif. Descrive in dettaglio tutti i passaggi necessari per far funzionare il progetto di esempio LVGL alif_m55-lvgl su Alif E7 Devkit Gen2; tuttavia, qualsiasi progetto basato sul Alif VS Code Template ha una procedura di installazione pressoché identica, quindi questa può essere utilizzata come guida generale per questi progetti. Esistono altri modi per compilare per le schede Alif, ad esempio con Zephyr RTOS. Vedere Alif's GitHub repos.

Questa guida è per Linux e Windows.

Questo progetto utilizza l'accelerazione di rendering 2D D/AVE con D/AVE 2D *draw unit* di LVGL.

Guida Passo-Passo

Installare Visual Studio Code

Installare Visual Studio code. Esistono diversi modi per installarlo a seconda della piattaforma. Vedere qui.

> **Nota**
>
> I passaggi rimanenti possono essere eseguiti **facoltativamente** all'interno di un container Docker. È possibile connettersi al container Docker come container di sviluppo remoto VS Code.
>
> Dovrebbero essere create due porte seriali al momento della connessione della scheda Alif. Su Linux, saranno simili a `/dev/ttyACM0` and `/dev/ttyACM1`. Nel comando `docker run` utilizzato per creare il container di sviluppo, includere `--device /dev/ttyACM0 --device /dev/ttyACM1` per consentire al container di accedere a quelle porte in modo da poter eseguire il flashing da esso.
>
> Installare l'estensione VS Code "Dev Containers". Selezionare il container da "Remote Explorer" nel pannello laterale sinistro.

Installare gli strumenti Prerequisiti

Assicurarsi che siano installati nel proprio ambiente. Le estensioni VS Code si basano sulla loro presenza.

- `git`
- `curl`
- `unzip`

Installare gli Alif SE Tool

Creare un account Alif e scaricare gli strumenti da here sotto "Alif Security Toolkit". Estrarre il file. Il path in cui è stato estratto sarà necessario in seguito.

Su Linux, l'estrazione può essere eseguita eseguendo quanto segue:

```
cd
tar -xf Downloads/APFW0002-app-release-exec-linux-SE_FW_1.101.00_DEV.tar
ls
pwd
```

Tra i risultati di `ls` si dovrebbe vedere `app-release-exec-linux`. Questo, combinato con l'output di `pwd`, è il percorso che si dovrà utilizzare in seguito. Ovvero, `/home/you/app-release-exec-linux`.

Installare il Software J-Link (facoltativo)

Scaricare l'ultima versione stabile del software J-Link Software. Il path di installazione sarà necessario in seguito.

Clonare il Progetto `alif_m55-lvgl`

```
git clone --recursive https://github.com/alifsemi/alif_m55-lvgl
```

Aprire `alif_m55-lvgl` in VS Code

Aprire il repository clonato in VS Code. Affinché le estensioni di VS Code funzionino correttamente, si consiglia di aprire la cartella in VS Code anziché aprire una directory padre contenente il progetto.

```
code alif_m55-lvgl
```

oppure si va su **File > Open Folder** in VS Code e si apre `alif_m55-lvgl`.

Se viene richiesto di installare automaticamente le estensioni consigliate, cliccare su "Install" per saltare il passaggio successivo.

Installare le Estensioni di VS Code Richieste

Installare le seguenti estensioni di VS Code dal pannello laterale "Extensions"

- Arm Tools Environment Manager
- Arm CMSIS Solution
- C/C++ Extension Pack
- Cortex-Debug (facoltativo necessario per il debug)

Activate Environment

Se non è avvenuto automaticamente, cliccare su "Arm Tools" nella barra in basso e poi su "Activate Environment" nell'elenco che appare. Verranno installati CMake, ninja-build, un compilatore GCC ARM e cmsis-toolbox.

Se si vede solo "Reactivate Environment", è probabile che sia già attivo.

Impostare i Path dei Tool Installati

Premere ctrl + shift + p. Digitare "preferences" e selezionare l'opzione "Preferences: Open User Settings (JSON)" tra le opzioni. Si aprirà un file `settings.json`. Nota: se si utilizza un container Docker, è meglio modificare il file JSON nella directory del progetto in `.vscode/settings.json`.

Si devono aggiungere alcune voci (almeno `"alif.setools.root"`) al JSON che si vede.

Se il `settings.json` inizialmente appare così...

```
{
    "workbench.colorTheme": "Default Dark+",
    "editor.renderWhitespace": "all",
}
```

... allora dovrebbe apparire così:

```
{
    "workbench.colorTheme": "Default Dark+",
    "editor.renderWhitespace": "all",
    "alif.setools.root" : "C:/alif-se-tools/app-release-exec",
    "cortex-debug.JLinkGDBServerPath": "C:/Program Files/SEGGER/JLink/
↪JLinkGDBServerCL.exe"
}
```

Quanto sopra usa i path Windows come esempio. Un path Linux per i Tool Alif SE potrebbe essere simile a `"/home/you/app-release-exec-linux"`.

Configurare la Variante della Board

Aprire il file board.h.

Identificare la variante della scheda nell'elenco e impostare `BOARD_ALIF_DEVKIT_VARIANT` sul valore corretto. Potrebbe essere necessario impostare anche `BOARD_ILI9806E_PANEL_VARIANT` se il valore predefinito non corrisponde al proprio.

Impostare il Build Context, Compilare e Flash-are

Accedere alla vista "Manage Solution" dall'estensione CMSIS Solution. Vi si può accedere cliccando sull'icona a forma di ingranaggio nella barra inferiore oppure andando al pannello CMSIS a sinistra e cliccando sull'ingranaggio in alto. Una volta aperto, si presenta come una scheda dell'editor grafico chiamata "Manage Solution" con un'icona a forma di ingranaggio.

Importante

In "Run and Debug" > "Run Configuration" (column) > "alif" (row), cliccare sul menù a discesa e selezionare "First time pack installation". Cliccare sull'icona di riproduzione nella parte superiore del pannello laterale sinistro di CMSIS. Nel terminale si dovrebbero vedere i pacchetti CMSIS in fase di installazione. Attendere il completamento.

Ora si può cliccare sull'icona del martello accanto all'icona di "play" per compilare il progetto. Verranno compilate alcune centinaia di file. Attendere il completamento.

Aprire il menù a discesa precedente e selezionare "Prepare program with Security Toolkit". Cliccare sull'icona di "play" [riproduzione]. Vengono preparati alcuni file come passaggio preliminare al flashing.

Aprire di nuovo il menu a discesa e selezionare "Program with Security Toolkit". Cliccare sull'icona di "play" [riproduzione]. Se è la prima volta, verrà chiesto di scegliere quale porta seriale utilizzare per flashare la scheda. Si può provare `/dev/ttyACM0`. Se è quella sbagliata, l'operazione fallirà e si dovrà aprire il menù a discesa e selezionare "Program with Security Toolkit (select COM port)" per sovrascrivere quella precedente che è stata salvata come default.

Il benchmark LVGL dovrebbe essere eseguito sulla scheda Alif al termine del flashing.

Ulteriori Informazioni

In caso di difficoltà durante la lettura di questa guida, fare riferimento a queste fonti Alif per passaggi più dettagliati.

- Getting Started with VSCode CMSIS pack project
- VSCode Getting Started Template

È possibile scaricare la "Alif Security Toolkit Quick Start Guide" da https://alifsemi.com/support/software-tools/ensemble/ , presupponendo che sia stato creato un account, per imparare a utilizzare gli strumenti Alif SE per eseguire manipolazioni di basso livello sulla scheda.

I Core HP e HE e Build Ottimizzata

Nella vista "Manage Solution" illustrata nella guida, è possibile selezionare un target HP o un target HE. Questi si riferiscono ai due core distinti presenti nell'Alif E7. "HE" sta per "High Efficiency" mentre "HP" sta per "High Performance". Per ottenere le migliori prestazioni da un'applicazione LVGL, selezionare HP. Considerare HE quando il consumo energetico è un problema. Il vantaggio di avere core asimmetrici è che l'applicazione può teoricamente eseguire carichi di lavoro a bassa priorità in modo efficiente sul core HE e delegare carichi di lavoro critici e ad alta intensità di elaborazione al core HP.

È anche possibile scegliere un "Build Type". Per prestazioni ottimali, scegliere "release". Se si esegue il debug, scegliere "debug".

Per massimizzare il punteggio nel benchmark LVGL e massimizzare le prestazioni di un'applicazione LVGL in generale, deve essere selezionato il core HP e il tipo di build dev'essere release.

4.4.2 Arm

Arm è un'azienda leader nella progettazione di semiconduttori e software, rinomata per aver creato i core dei microcontrollori (MCU) Cortex-M e quelli dei processori Cortex-A/R (MPU), che sono parte integrante di un'ampia gamma di dispositivi. Questi core sono il cuore di molti sistemi embedded e alimentano chip di giganti del settore come STMicroelectronics, NXP e Renesas. I progetti a basso consumo energetico di Arm sono utilizzati in miliardi di dispositivi in tutto il mondo, dai microcontrollori agli smartphone e ai server. Concedendo in licenza i propri progetti di processori, Arm consente a un ampio ecosistema di partner di sviluppare soluzioni personalizzate ottimizzate per prestazioni, consumo energetico e dimensioni. L'architettura di Arm è altamente compatibile con vari sistemi operativi e librerie software, tra cui LVGL, rendendola una scelta versatile per gli sviluppatori che creano interfacce utente grafiche efficienti e ad alte prestazioni.

Compilare LVGL per Arm

Non è richiesta alcuna azione specifica. Qualsiasi compilatore che supporti l'architettura Arm come target può essere utilizzato per compilare il codice sorgente di LVGL, inclusi GCC, LLVM e AC6.

È anche possibile cross-compilare LVGL per una MPU (invece di compilarlo sull'hardware target) o creare una libreria shared. Per ulteriori informazioni, consultare *CMake*.

Introduzione a AC6

Poiché AC6 è una toolchain proprietaria, contiene molte ottimizzazioni specifiche, quindi è possibile aspettarsi le migliori prestazioni quando lo si utilizza.

AC6 non è gratuito, ma offre una licenza community che può essere attivata come segue:

1. Scaricare e installare il compilatore AC6 dal sito web di Arm.
2. Per registrare una licenza community, andare alla cartella `bin` del compilatore e, in un terminale, eseguire `armlm.exe activate -server https://mdk-preview.keil.arm.com -product KEMDK-COM0` (Su Linux, usare `./armlm`).

Supporto IDE

Non ci sono limitazioni sugli IDE supportati. LVGL funziona negli IDE di vari fornitori, tra cui Keil MDK di Arm, IAR, e2 studio di Renesas, MCUXpresso di NXP, CubeIDE di ST, nonché progetti custom make o CMake.

Arm2D e il set di istruzioni Helium

Arm Cortex-M55 e Cortex-M85 dispongono del set di istruzioni SIMD Helium. Tra le altre cose, questo può effettivamente velocizzare il rendering dell'interfaccia utente. *Arm2D* è una libreria gestita da Arm che sfrutta il set di istruzioni Helium.

Notare che GCC presenta alcuni problemi noti con le "Helium intrinsics". Si consiglia di utilizzare AC6 o LLVM quando si utilizza codice Helium.

Per aggiungere Arm2D al progetto, seguire questi passaggi:

1. Per sfruttarne la potenza, assicurarsi che `mcpu` sia impostato su `cortex-m85`, `cortex-m55`, o `cortex-m52` e aggiungere il flag `-fvectorize`. Per testare senza SIMD, si usa ad esempio `cortex-m85+nomve`.
2. Arm2D può essere scaricato da https://github.com/ARM-software/Arm-2D. Valutare la possibilità di utilizzare il branch `developing`, che contiene gli aggiornamenti più recenti.
3. Aggiungere `Arm-2D/Library/Include` ai path degli "include".
4. Copiare `Arm-2D/Library/Include/template/arm_2d_cfg.h` nella posizione che si preferisce per fornire la configurazione di default per Arm2D. La cartella che contiene `arm_2d_cfg.h` dev'essere aggiunta al path dell'include.
5. Il repository Arm2D contiene diversi esempi e template; tuttavia, assicurarsi che venga compilato solo `Arm-2D/Library/Source`.

6. Anche la libreria CMSIS DSP deve essere aggiunta al progetto. Si può utilizzare CMSIS-PACKS o aggiungerla manualmente.
7. Per prestazioni migliori, abilitare `LTO` (Link Time Optimization) e usare `-Omax` o `-Ofast`.
8. Arm2D tenta di leggere/scrivere più dati con una singola istruzione. Pertanto, è importante utilizzare la memoria più veloce (ad esempio, `BSS` o `TCM`) per il buffer di LVGL per evitare colli di bottiglia nella larghezza di banda della memoria.
9. Abilitare `LV_USE_DRAW_ARM2D_SYNC 1` e `LV_USE_DRAW_SW_ASM LV_DRAW_SW_ASM_HELIUM` in `lv_conf.h`.

Accelerazione Neon

Diversi microprocessori Cortex-A supportano il set di istruzioni Neon SIMD. LVGL ha un supporto integrato per migliorare le prestazioni del rendering software utilizzando le istruzioni Neon. Per abilitare l'accelerazione Neon, impostare `LV_USE_DRAW_SW_ASM` to `LV_DRAW_SW_ASM_NEON` in `lv_conf.h`.

4.4.3 Espressif (Serie di Chip ESP32)

LVGL può essere utilizzato e configurato come componente standard ESP-IDF.

Se non si ha familiarità con ESP-IDF, seguire le istruzioni nella ESP-IDF Programming guide per installare e configurare ESP-IDF sul computer.

Progetti demo LVGL per ESP32

Per un rapido avvio con LVGL ed ESP32, sono disponibili i seguenti progetti demo preconfigurati per schede di sviluppo specifiche:

- ESP-BOX-3
- ESP32-S3-LCD-EV-BOARD
- M5Stack-CoreS3

Fare riferimento ai file README.md in questi repository per le istruzioni di compilazione e flashing.

Questi progetti demo utilizzano i Board Support Package (BSP) di Espressif. Ulteriori BSP ed esempi sono disponibili nel repository esp-bsp.

Utilizzo di LVGL nel proprio progetto ESP-IDF

Il modo più semplice per integrare LVGL nel proprio progetto ESP-IDF è tramite il componente esp_lvgl_port. Questo componente, utilizzato nei progetti demo menzionati sopra, fornisce funzioni di supporto per una facile installazione di LVGL e driver di display. Inoltre, può aggiungere il supporto per touch, encoder rotativi, pulsanti o ingressi USB HID. Semplifica il risparmio energetico, la rotazione dello schermo e altre funzionalità specifiche della piattaforma.

esp_lvgl_port supporta le versioni LVGL 8 e 9 ed è compatibile con ESP-IDF v4.4 e successive. Per aggiungerlo al progetto, si usa il seguente comando:

```
idf.py add-dependency "espressif/esp_lvgl_port^2.3.0"
```

Per default, esp_lvgl_port dipende dall'ultima versione stabile di LVGL, quindi non sono necessari passaggi aggiuntivi per i nuovi progetti. Se è richiesta una versione specifica di LVGL, specificatela nel progetto per evitare aggiornamenti automatici. LVGL può essere utilizzato anche senza esp_lvgl_port, come descritto di seguito.

Ottenere LVGL

LVGL è distribuito tramite ESP Registry, dove vengono caricate tutte le release LVGL. Se non si desidera utilizzare esp_lvgl_port, è possibile aggiungere LVGL component al progetto con il seguente comando:

```
idf.py add-dependency "lvgl/lvgl^9.*"
```

Modificare la parte `^9.*` in base ai requisiti di versione LVGL. Ulteriori informazioni sulle specifiche di versione sono disponibili nella documentazione di IDF Component Manager. Durante la build successiva, il componente LVGL verrà recuperato dal registro dei componenti e aggiunto al progetto.

Uso avanzato: Utilizzare LVGL come componente locale

Per lo sviluppo e il test di LVGL, potrebbe essere utile utilizzare LVGL come componente locale anziché dal Registro ESP, che offre solo versioni rilasciate e non consente modifiche locali. Per farlo, clonare LVGL nel proprio progetto con il seguente comando:

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

> **Nota**
>
> Tutti i componenti da `${project_dir}/components` vengono aggiunti automaticamente alla build.

Configurazione

Per configurare LVGL, avviare il menù di configurazione con `idf.py menuconfig` nella directory principale del progetto. Accedi a `Component config` e poi a `LVGL configuration`.

Supporto per Driver di Display e Touch

Per un progetto LVGL di successo, ci sarà bisogno di un driver display e, facoltativamente, di un driver touch. Espressif fornisce questi driver basati sul suo componente esp_lcd.

- esp_lcd supporta nativamente alcuni display di base
- Altri display sono mantenuti nel repository esp-bsp repository e vengono caricati nell'ESP Registry
- I driver touch sono mantenuti nel repository esp-bsp repository e vengono caricati nell'ESP Registry

Questi componenti condividono un'API pubblica comune, semplificando la migrazione dei progetti tra diversi driver di display e touch.

Per aggiungere un driver di display o touch al progetto, utilizzare un comando come:

```
idf.py add-dependency "espressif/esp_lcd_gc9a01^2.0.0"
```

Utilizzo del file system in ESP-IDF

ESP-IDF utilizza le funzioni standard C per le operazioni sui file (`fopen`, `fread`) in tutte le sue API relative all'archiviazione. Ciò consente un'interoperabilità perfetta con LVGL quando si abilita la configurazione *`LV_USE_FS_STDIO`*. Il processo è descritto in dettaglio di seguito, utilizzando `SPIFFS` come dimostrazione.

- **Decidere quale sistema di archiviazione utilizzare**

  ESP-IDF offre molti esempi pronti all'uso, come SPIFFS , SD Card e LittleFS .

- **Riconfigurare il progetto**

  Il progetto di esempio dovrebbe essere esaminato per i dettagli, ma in generale le modifiche riguardano:

  - Abilitazione del file system STDIO di LVGL nella configurazione

    È possibile utilizzare `menuconfig`:

    * `Component config → LVGL configuration → 3rd Party Libraries`: enable `File system on top of stdio API`

    * Poi selezionare `Set an upper cased letter on which the drive will accessible` e impostarla su `65` (ASCII **A**)
    * È anche possibile impostare `Default driver letter` a 65 per ignorare il prefisso nei path dei file.

  – Modifica della tabella delle partizioni

    La configurazione esatta dipende dalle dimensioni della flash e dalle partizioni esistenti, ma il nuovo risultato finale dovrebbe essere simile a questo:

Tabella 1: Partition Table

nvs	data	nvs	0x9000	0x6000
phy_init	data	phy	0xf000	0x1000
factory	app	factory	0x10000	1400k
storage	data	spiffs		400k

> **Nota**
>
> Se non si utilizza ancora un file `partition.csv` personalizzato, è possibile aggiungerlo tramite `menuconfig` (`Partition Table → Partition Table → Custom partition table CSV`).

  – Applicare le modifiche al sistema di build

    Alcuni file system ESP forniscono la generazione automatica da una cartella host tramite CMake. Le righe corrette devono essere copiate in `main/CMakeLists.txt`

> **Nota**
>
> `LittleFS` ha dipendenze aggiuntive che devono essere aggiunte a `main/idf_component.yml`

- **Preparare i file delle immagini**

  Il tool Python `LVGLImage.py` di LVGL può essere utilizzato per convertire le immagini in file binari di pixel map. Supporta vari formati e compressioni.

  Mentre librerie di terze parti (come *LodePNG* e *Tiny JPEG*) consentono di utilizzare file immagine senza conversione.

  Dopo aver preparato i file, è necessario spostarli sul dispositivo di destinazione:

  – Se attivato correttamente, un file system **SPIFFS** basato sulla cartella `spiffs_image` dovrebbe essere generato automaticamente e successivamente flashato sul dispositivo di destinazione.

  – Un meccanismo simile per **LittleFS** usa la cartella `flash_data`, ma è disponibile solo per host Linux.

  – Per la **SD Card**, è possibile utilizzare un file browser tradizionale.

- *Invoca le chiamate API appropriate nel codice dell'applicazione**

  Le funzionalità principali richiedono solo poche righe. L'esempio seguente disegna anche l'immagine.

```
#include "esp_spiffs.h"

void lv_example_image_from_esp_fs(void) {
```

(continues on next page)

(continua dalla pagina precedente)

```
    esp_vfs_spiffs_conf_t conf = {
        .base_path = "/spiffs",
        .partition_label = NULL,
        .max_files = 5,
        .format_if_mount_failed = false
    };

    esp_err_t ret = esp_vfs_spiffs_register(&conf);

    if (ret != ESP_OK) {
        ESP_LOGE(TAG, "Failed to register SPIFF filesystem");
        return;
    }

    lv_obj_t * obj = lv_image_create(lv_screen_active());
    lv_image_set_src(widget, "A:/spiffs/logo.bin");
    lv_obj_center(widget);
}
```

- **Build e flash**

  Dopo aver chiamato `idf.py build flash`, l'immagine dovrebbe essere visualizzata sullo schermo.

> **Nota**
>
> Le modifiche apportate da `menuconfig` non vengono tracciate nel repository se il file `sdkconfig` viene aggiunto a `.gitignore`, che è l'impostazione di default per molti progetti ESP-IDF. Per rendere permanente la configurazione, aggiungere le seguenti righe a `sdkconfig.defaults`:
>
> ```
> CONFIG_PARTITION_TABLE_CUSTOM=y
> CONFIG_LV_USE_FS_STDIO=y
> CONFIG_LV_FS_STDIO_LETTER=65
> CONFIG_LV_FS_DEFAULT_DRIVER_LETTER=65
> ```

Supporto per Pixel Processing Accelerator

Alcune serie di chip ESP32, come l'ESP32-P4, supportano l'hardware Pixel Processing Accelerator (PPA), in grado di accelerare le operazioni di riempimento e fusione delle immagini. Questa periferica funziona con l'hardware DMA-2D, responsabile dello spostamento dei buffer di input/output da/verso il motore di elaborazione PPA.

Dispositivi supportati

I target Espressif che supportano il PPA sono:

- Le serie ESP32-P4.

Utilizzo del PPA nel progetto ESP-IDF

LVGL supporta, a livello sperimentale, l'accelerazione del riempimento e della fusione delle immagini tramite PPA. L'utente può abilitarla all'interno del proprio `sdkconfig.default` aggiungendo la seguente opzione per abilitare l'unità di disegno PPA insieme al rendering software:

```
CONFIG_LV_USE_PPA=y
```

Salvare il file e quindi ricostruire il progetto. Sarà sufficiente aggiungere il codice PPA e l'esecuzione inizierà automaticamente, quindi non sono necessari ulteriori passaggi dal punto di vista del codice utente.

Limitazioni

Si noti che il PPA è a livello sperimentale, dove sono previsti alcuni miglioramenti delle prestazioni nelle attività di disegno relative alla copia o al riempimento di rettangoli, mentre per la fusione delle immagini, sebbene operativa, non si registrano miglioramenti significativi; la causa iniziale, secondo la sezione PPA del manuale di riferimento, è dovuta alla larghezza di banda della memoria DMA-2D.

4.4.4 NXP

NXP ha integrato LVGL nei pacchetti "SDK MCUXpresso" per diversi dei nostri microcontrollori come componente software opzionale, consentendo una facile valutazione e migrazione nella progettazione del prodotto. LVGL è una libreria grafica embedded gratuita e open source con funzionalità che consentono di creare GUI embedded con elementi grafici intuitivi, splendidi effetti visivi e un ridotto utilizzo di memoria. Il framework grafico completo include una varietà di widget da utilizzare nella creazione della GUI e supporta funzioni più avanzate come animazioni e anti-aliasing.

LVGL abilita la grafica nel nostro tool gratuito "GUI Guider". È disponibile per l'uso con i microcontrollori generici e crossover di NXP, fornendo agli sviluppatori uno strumento per creare applicazioni GUI complete e di alta qualità con LVGL.

Creazione di un nuovo progetto con LVGL

Scaricare subito un SDK per una board supportata e iniziare a sviluppare la prossima applicazione GUI. È completamente configurato con LVGL (e con supporto PXP/VGLite/G2D se i moduli sono presenti), non è richiesto alcun ulteriore lavoro di integrazione.

Accelerazione HW per piattaforme NXP iMX RT

A seconda della piattaforma RT utilizzata, l'accelerazione può essere eseguita da NXP PXP (PiXel Pipeline) e/o dalla GPU Verisilicon tramite un'API denominata VGLite. Ogni acceleratore ha un proprio contesto che ne consente l'utilizzo sia singolarmente che simultaneamente (in modalità multithreading LVGL).

Accelerazione HW per piattaforme NXP iMX

Sulle piattaforme MPU, l'accelerazione può essere eseguita (indipendentemente dall'hardware) dalla libreria NXP G2D. Questo acceleratore ha un proprio contesto che ne consente l'utilizzo sia singolarmente che simultaneamente con la CPU (in modalità multithreading LVGL).

Acceleratore PXP

Configurazione di base:

- Selezionare il motore PXP NXP in "lv_conf.h": Impostare *LV_USE_PXP* a `1`.
- Per utilizzare PXP come unità di disegno, selezionare in "lv_conf.h": Impostare `LV_USE_DRAW_PXP` a `1`.
- Per utilizzare PXP per ruotare lo schermo, selezionare in "lv_conf.h": Impostare `LV_USE_ROTATE_PXP` a `1`.
- Abilitare le asserzioni PXP in "lv_conf.h": Impostare :c:macro: `LV_USE_PXP_ASSERT` a `1`. Esistono alcune asserzioni PXP che possono interrompere l'esecuzione del programma nel caso in cuiEsistono alcune asserzioni PXP che possono interrompere l'esecuzione del programma nel caso in cui c:macro:`LV_ASSERT_HANDLER` sia impostato su `while(1);` (Halt per default). In caso contrario, verrà registrato solo un messaggio di errore tramite `LV_LOG_ERROR`.
- Se è definito il simbolo `SDK_OS_FREE_RTOS`, verrà utilizzata l'implementazione di FreeRTOS, altrimenti verrà incluso codice bare metal.

Inizializzazione di base:

L'inizializzazione del disegno PXP viene eseguita automaticamente in `lv_init()` once the PXP is enabled as a draw unit or to rotate the screen, no user code is required:

```
#if LV_USE_DRAW_PXP || LV_USE_ROTATE_PXP
  lv_draw_pxp_init();
#endif
```

Durante l'inizializzazione del PXP, verrà creata una nuova unità di disegno `lv_draw_pxp_unit_t` con le callback aggiuntive, se `LV_USE_DRAW_PXP` è impostato a `1`:

```
lv_draw_pxp_unit_t * draw_pxp_unit = lv_draw_create_unit(sizeof(lv_draw_
↪pxp_unit_t));
draw_pxp_unit->base_unit.evaluate_cb = _pxp_evaluate;
draw_pxp_unit->base_unit.dispatch_cb = _pxp_dispatch;
draw_pxp_unit->base_unit.delete_cb = _pxp_delete;
```

e verrà generato un thread aggiuntivo `_pxp_render_thread_cb()` per gestire i task di disegno supportati.

```
#if LV_USE_PXP_DRAW_THREAD
    lv_thread_init(&draw_pxp_unit->thread, "pxpdraw", LV_THREAD_PRIO_HIGH,␣
↪_pxp_render_thread_cb, 2 * 1024, draw_pxp_unit);
#endif
```

Se `LV_USE_PXP_DRAW_THREAD` non è definito, non verrà creato alcun thread di disegno aggiuntivo e il task di disegno PXP verrà eseguito sullo stesso thread principale LVGL.

`_pxp_evaluate()` verrà chiamato dopo la creazione di ogni attività e analizzerà se l'attività è supportata da PXP o meno. Se supportato, all'attività verranno assegnati un punteggio preferito e l'ID dell'unità di disegno. Uno `score` uguale a `100` è lo score di default della CPU. Uno score più piccolo indica che PXP è in grado di disegnarlo più velocemente.

`_pxp_dispatch()` è la callback del dispatcher PXP, che accetta un task pronto per il disegno (con `DRAW_UNIT_ID_PXP` impostato) e la passa all'unità di disegno PXP per l'elaborazione.

`_pxp_delete()` pulisce l'unità di disegno PXP.

Funzionalità supportate:

Diverse funzionalità di disegno in LVGL possono essere demandate al motore PXP. La CPU è disponibile per altre operazioni mentre PXP è in esecuzione. È necessario un RTOS per bloccare il thread di disegno LVGL e passare a un altro task o sospendere la CPU per risparmiare energia.

Le attività di disegno supportate sono disponibili in "src/draw/nxp/pxp/lv_draw_pxp.c":

```
switch(t->type) {
    case LV_DRAW_TASK_TYPE_FILL:
        lv_draw_pxp_fill(t, t->draw_dsc, &t->area);
        break;
    case LV_DRAW_TASK_TYPE_IMAGE:
        lv_draw_pxp_img(t, t->draw_dsc, &t->area);
        break;
    case LV_DRAW_TASK_TYPE_LAYER:
        lv_draw_pxp_layer(t, t->draw_dsc, &t->area);
        break;
    default:
        break;
}
```

Inoltre, la rotazione dello schermo può essere gestita da PXP:

- Riempire l'area con un colore (senza raggio, senza gradiente) + opacità opzionale.
- Blit [Block Image Transfer] dell'immagine sorgente RGB565/ARGB888/XRGB8888 sulla destinazione. RGB565/RGB888/ARGB888/XRGB8888 + opacità opzionale.
- Ricolorare l'immagine sorgente RGB565.
- Ridimensionare e ruotare (90, 180, 270 gradi) l'immagine sorgente RGB565.
- Fusione dei livelli (con gli stessi formati supportati per il blitting).
- Ruotare lo schermo (90, 180, 270 gradi).

Limitazioni note:

- PXP può ruotare i fotogrammi solo di angoli multipli di 90 gradi.
- La rotazione non è supportata per le immagini non allineate a blocchi di 16x16 pixel. PXP è impostato per elaborare blocchi 16x16 per ottimizzare il sistema in termini di larghezza di banda di memoria e tempo di elaborazione delle immagini. Il motore di output essenzialmente tronca tutti i pixel di output dopo che è stato scritto il numero desiderato di pixel. Quando si ruota un'immagine sorgente e l'output non è divisibile per la dimensione del blocco, i pixel errati potrebbero essere troncati e l'immagine di output finale potrebbe apparire spostata.
- La ricolorazione o la trasformazione di immagini con opacità o canale alfa non possono essere ottenute in una singola configurazione della pipeline PXP. Sarebbero necessari due o più passaggi.
- L'indirizzo del buffer deve essere allineato a 64 byte: impostare `LV_DRAW_BUF_ALIGN` a `64` in "lv_conf.h". Non è richiesto alcun allineamento di passo: impostare `LV_DRAW_BUF_STRIDE_ALIGN` a `1` in "lv_conf.h".

Setup del progetto:

- Aggiungere i file sorgente relativi a PXP (e i relativi header, se disponibili) al progetto:
  - "src/draw/nxp/pxp/lv_draw_buf_pxp.c": callback del buffer di disegno
  - "src/draw/nxp/pxp/lv_draw_pxp_fill.c": area di riempimento
  - "src/draw/nxp/pxp/lv_draw_pxp_img.c": immagine blit (con ricolorazione o trasformazione opzionale)
  - "src/draw/nxp/pxp/lv_draw_pxp_layer.c": fusione dei livelli
  - "src/draw/nxp/pxp/lv_draw_pxp.c": inizializzazione dell'unità di disegno
  - "src/draw/nxp/pxp/lv_pxp_cfg.c": init, deinit, run/wait del dispositivo PXP
  - "src/draw/nxp/pxp/lv_pxp_osa.c": astrazione del sistema operativo (FreeRTOS o bare metal)
  - "src/draw/nxp/pxp/lv_pxp_utils.c": helper delle funzioni
- Il codice relativo a PXP dipende da due driver forniti dall'SDK MCU. Questi driver devono essere aggiunti al progetto:
  - fsl_pxp.c: Driver PXP
  - fsl_cache.c: Funzioni di gestione della cache della CPU

Configurazione di default di PXP:

- L'implementazione dipende da più funzioni specifiche del sistema operativo. La struttura `pxp_cfg_t` con puntatori di callback viene utilizzata come parametro per la funzione `lv_pxp_init()`. Implementazione di default per FreeRTOS in lv_pxp_osa.c.
  - `pxp_interrupt_init()`: Inizializza l'interrupt PXP (configurazione HW, configurazione del sistema operativo)
  - `pxp_interrupt_deinit()`: Deinizializza l'interrupt PXP (configurazione HW, configurazione del sistema operativo)

- `pxp_run()`: Avvia il processo PXP. Utilizza un meccanismo specifico del sistema operativo per bloccare il thread di disegno.
- `pxp_wait()`: Attende il completamento di PXP.

Acceleratore VGLite

Il motore VGLite può gestire funzionalità di disegno aggiuntive in LVGL. La CPU è disponibile per altre operazioni mentre VGLite è in esecuzione. È necessario un RTOS per bloccare il thread di disegno LVGL e passare a un altro task o sospendere la CPU per risparmiare energia.

Configurazione di base:

- Selezionare il motore NXP VGLite in "lv_conf.h": Impostare *LV_USE_DRAW_VGLITE* a `1`. Il simbolo `SDK_OS_FREE_RTOS` deve essere definito in modo che l'implementazione "osal" [OS Abstraction Layer] del driver FreeRTOS sia abilitata.
- Abilitare le asserzioni VGLite in "lv_conf.h": Impostare :c:macro: `LV_USE_VGLITE_ASSERT` a `1`. Le asserzioni VGLite verificheranno il codice di stato dell'API del driver e, in caso di errore, potranno interrompere l'esecuzione del programma nel caso in cui c:macro: `LV_ASSERT_HANDLER` sia impostato su `while(1);` (Halt per default). In caso contrario, verrà registrato solo un messaggio di errore tramite `LV_LOG_ERROR`.

Inizializzazione di base:

Inizializza la GPU VGLite prima di chiamare *lv_init()* specificando la larghezza/altezza della finestra di tassellatura. I valori di default per la larghezza e l'altezza della tassellatura e la dimensione del buffer dei comandi si trovano nel file SDK "vglite_support.h".

```
#if LV_USE_DRAW_VGLITE
    #include "vg_lite.h"
    #include "vglite_support.h"
#endif
...
#if LV_USE_DRAW_VGLITE
if(vg_lite_init(DEFAULT_VG_LITE_TW_WIDTH, DEFAULT_VG_LITE_TW_HEIGHT) != VG_
↪LITE_SUCCESS)
{
    PRINTF("VGLite init error. STOP.");
    vg_lite_close();
    while (1)
        ;
}

if (vg_lite_set_command_buffer_size(VG_LITE_COMMAND_BUFFER_SIZE) != VG_
↪LITE_SUCCESS)
{
    PRINTF("VGLite set command buffer. STOP.");
    vg_lite_close();
    while (1)
        ;
}
#endif
```

L'inizializzazione del disegno di VGLite viene eseguita automaticamente in *lv_init()* una volta abilitato VGLite, non è richiesto alcun codice utente:

```
#if LV_USE_DRAW_VGLITE
    lv_draw_vglite_init();
#endif
```

Durante l'inizializzazione di VGLite, verrà creata una nuova unità di disegno `lv_draw_vglite_unit_t` con le callback aggiuntive:

```
lv_draw_vglite_unit_t * draw_vglite_unit = lv_draw_create_unit(sizeof(lv_
↪draw_vglite_unit_t));
draw_vglite_unit->base_unit.evaluate_cb = _vglite_evaluate;
draw_vglite_unit->base_unit.dispatch_cb = _vglite_dispatch;
draw_vglite_unit->base_unit.delete_cb = _vglite_delete;
```

e verrà generato un thread aggiuntivo `_vglite_render_thread_cb()` per gestire le attività di disegno supportate.

```
#if LV_USE_VGLITE_DRAW_THREAD
    lv_thread_init(&draw_vglite_unit->thread, "vglitedraw", LV_THREAD_PRIO_
↪HIGH, _vglite_render_thread_cb, 2 * 1024, draw_vglite_unit);
#endif
```

Se `LV_USE_VGLITE_DRAW_THREAD` non è definito, non verrà creato alcun thread di disegno aggiuntivo e l'attività di disegno di VGLite verrà eseguita sullo stesso thread principale di LVGL.

`_vglite_evaluate()` verrà chiamato dopo la creazione di ogni attività e analizzerà se l'attività è supportata da VGLite o meno. Se supportato, all'attività verranno assegnati un punteggio preferito e l'ID dell'unità di disegno. Uno `score` uguale a `100` è lo score di default della CPU. Uno score più piccolo indica che VGLite è in grado di disegnarlo più velocemente.

`_vglite_dispatch()` è la callback del dispatcher VGLite, che accetta un task pronto per il disegno (con `DRAW_UNIT_ID_VGLITE` impostato) e la passa all'unità di disegno VGLite per l'elaborazione.

`_vglite_delete()` pulirà l'unità di disegno VGLite.

Configurazione avanzata:

- Abilitare la divisione blit VGLite in "lv_conf.h": Impostare :c:macro: `LV_USE_VGLITE_BLIT_SPLIT` a `1`. Abilitando la soluzione alternativa per la divisione blit si attenuerà qualsiasi problema di degradazione della qualità su schermi con dimensioni superiori a 352 pixel.

  ```
  #define VGLITE_BLIT_SPLIT_THR 352
  ```

- Per default, la soglia del "blit split" è impostata su 352. I blit [Block Image Transfer] con larghezza o altezza superiori a questo valore verranno eseguiti in più passaggi. Il valore deve essere un multiplo dell'allineamento "stride"' in px. Per la maggior parte dei formati colore, l'allineamento è di 16 px (ad eccezione dei formati indice). La trasformazione non sarà supportata una volta con la divisione blit.
- Abilitare l'attività di disegno VGLite in modo sincrono in "lv_conf.h": Impostare :c:macro: `LV_USE_VGLITE_DRAW_ASYNC` a `1`. È possibile mettere in coda più task di disegno e scaricarle una volta sulla GPU in base allo stato di inattività della GPU. Se la GPU è occupata, l'attività verrà messa in coda e il dispatcher VGLite richiederà una nuova attività disponibile. Se la GPU è inattiva, la coda con eventuali attività in sospeso verrà scaricata sulla GPU. Lo stato di completamento dell'attività di disegno verrà inviato al thread LVGL principale in modo asincrono.

Funzionalità supportate:

Diverse funzionalità di disegno in LVGL possono essere scaricate sul motore VGLite. La CPU è disponibile per altre operazioni mentre la GPU è in esecuzione. È necessario un RTOS per bloccare il thread di disegno LVGL e passare a un'altra attività o sospendere la CPU per risparmiare energia.

Le attività di disegno supportate sono disponibili in "src/draw/nxp/vglite/lv_draw_vglite.c":

```
switch(t->type) {
    case LV_DRAW_TASK_TYPE_LABEL:
        lv_draw_vglite_label(t, t->draw_dsc, &t->area);
```

(continues on next page)

(continua dalla pagina precedente)

```
        break;
    case LV_DRAW_TASK_TYPE_FILL:
        lv_draw_vglite_fill(t, t->draw_dsc, &t->area);
        break;
    case LV_DRAW_TASK_TYPE_BORDER:
        lv_draw_vglite_border(t, t->draw_dsc, &t->area);
        break;
    case LV_DRAW_TASK_TYPE_IMAGE:
        lv_draw_vglite_img(t, t->draw_dsc, &t->area);
        break;
    case LV_DRAW_TASK_TYPE_ARC:
        lv_draw_vglite_arc(t, t->draw_dsc, &t->area);
        break;
    case LV_DRAW_TASK_TYPE_LINE:
        lv_draw_vglite_line(t, t->draw_dsc);
        break;
    case LV_DRAW_TASK_TYPE_LAYER:
        lv_draw_vglite_layer(t, t->draw_dsc, &t->area);
        break;
    case LV_DRAW_TASK_TYPE_TRIANGLE:
        lv_draw_vglite_triangle(t, t->draw_dsc);
        break;
    default:
        break;
}
```

Tutte le operazioni seguenti possono essere eseguite anche con opacità opzionale.

- Riempire l'area con il colore (con raggio o gradiente).
- Eseguire il blit [Block Image Transfer] dell'immagine sorgente (qualsiasi formato da `_vglite_src_cf_supported()`) sulla destinazione (qualsiasi formato da `_vglite_dest_cf_supported()`).
- Ricolorare l'immagine sorgente.
- Ridimensionare e ruotare (90, 180, 270 gradi) l'immagine sorgente RGB565.
- Fusione dei livelli (con gli stessi formati supportati per il blitting).
- Disegnare lettere (lettere bitmap blit / font raster).
- Disegnare bordi completi (LV_BORDER_SIDE_FULL).
- Disegnare archi (con bordi arrotondati).
- Disegnare linee (con trattini o bordi arrotondati).
- Disegnare triangoli colorati (con gradiente).

Limitazioni note:

- Allineamento dell'immagine sorgente: Il requisito di allineamento dei byte per un pixel dipende dal formato specifico del pixel. Sia l'indirizzo del buffer che il passo del buffer devono essere allineati. Come regola generale, l'allineamento è impostato su 16 pixel. Questo fa sì che l'allineamento dell'indirizzo del buffer sia di 32 byte per RGB565 e di 64 byte per ARGB8888.
- Per la destinazione del pixel engine (PE), l'allineamento dovrebbe essere di 64 byte per tutti i layout di buffer a mosaico (4x4). Il pixel engine non ha requisiti di allineamento aggiuntivi per i layout di buffer lineari (`VG_LITE_LINEAR`).

Setup del progetto:

- Aggiungere i file sorgente correlati a VGLite (e lgli header corrispondenti, se disponibili) al progetto:
  - "src/draw/nxp/vglite/lv_draw_buf_vglite.c": callback del buffer di disegno
  - "src/draw/nxp/vglite/lv_draw_vglite_arc.c": disegno dell'arco
  - "src/draw/nxp/vglite/lv_draw_vglite_border.c": disegno del bordo
  - "src/draw/nxp/vglite/lv_draw_vglite_fill.c": riempimento dell'area
  - "src/draw/nxp/vglite/lv_draw_vglite_img.c": immagine blit (con ricolorazione o trasformazione facoltativa)
  - "src/draw/nxp/vglite/lv_draw_vglite_label.c": disegno dell'etichetta
  - "src/draw/nxp/vglite/lv_draw_vglite_layer.c": fusione livelli
  - "src/draw/nxp/vglite/lv_draw_vglite_line.c": disegno linee
  - "src/draw/nxp/vglite/lv_draw_vglite_triangle.c": disegna triangoli
  - "src/draw/nxp/vglite/lv_draw_vglite.c": inizializzazione dell'unità di disegno
  - "src/draw/nxp/vglite/lv_vglite_buf.c": init/get buffer vglite
  - "src/draw/nxp/vglite/lv_vglite_matrix.c": imposta la matrice vglite
  - "src/draw/nxp/vglite/lv_vglite_path.c": crea i dati del percorso vglite
  - "src/draw/nxp/vglite/lv_vglite_utils.c": helper delle funzioni

Acceleratore G2D

Configurazione di base:

- Selezionare il motore NXP G2D in "lv_conf.h": Impostare `LV_USE_G2D` a `1`.
- Per utilizzare G2D come unità di disegno, selezionare in "lv_conf.h": Impostare *`LV_USE_DRAW_G2D`* a `1`.
- Abilitare le asserzioni G2D in "lv_conf.h": Impostare :c:macro: `LV_USE_G2D_ASSERT` a `1`. Esistono alcune asserzioni G2D che possono interrompere l'esecuzione del programma nel caso in cui c:macro: `LV_ASSERT_HANDLER` sia impostato su `while(1);` (Halt per default). In caso contrario, verrà registrato solo un messaggio di errore tramite `LV_LOG_ERROR`.

Inizializzazione di base:

L'inizializzazione del disegno G2D viene eseguita automaticamente in *`lv_init()`* una volta che G2D è abilitato come unità di disegno, non è richiesto alcun codice utente:

```
#if LV_USE_DRAW_G2D
  lv_draw_g2d_init();
#endif
```

Durante l'inizializzazione di G2D, verrà creata una nuova unità di disegno `lv_draw_g2d_unit_t` con le callback aggiuntive, se *`LV_USE_DRAW_G2D`* è impostato a `1`:

```
lv_draw_g2d_unit_t * draw_g2d_unit = lv_draw_create_unit(sizeof(lv_draw_
↪g2d_unit_t));
draw_g2d_unit->base_unit.evaluate_cb = _g2d_evaluate;
draw_g2d_unit->base_unit.dispatch_cb = _g2d_dispatch;
draw_g2d_unit->base_unit.delete_cb = _g2d_delete;
```

e verrà generato un thread aggiuntivo `_g2d_render_thread_cb()` per gestire le attività di disegno supportate.

```
#if LV_USE_G2D_DRAW_THREAD
  lv_thread_init(&draw_g2d_unit->thread, LV_THREAD_PRIO_HIGH, _g2d_render_
↪thread_cb, 2 * 1024, draw_g2d_unit);
#endif
```

Se `LV_USE_G2D_DRAW_THREAD` non è definito, non verrà creato alcun thread di disegno aggiuntivo e l'attività di disegno G2D verrà eseguita sullo stesso thread principale LVGL.

`_g2d_evaluate()` verrà chiamato dopo la creazione di ogni attività e analizzerà se l'attività è supportata da G2D o meno. Se supportato, all'attività verranno assegnati un punteggio preferito e l'ID dell'unità di disegno. Uno `score` uguale a `100` è lo score di default della CPU. Uno score più piccolo indica che G2D è in grado di disegnarlo più velocemente.

`_g2d_dispatch()` è la callback del dispatcher G2D, che prenderà un'attività pronta per essere disegnata (con `DRAW_UNIT_ID_G2D` impostato) e la passerà all'unità di disegno G2D per l'elaborazione.

`_g2d_delete()` pulirà l'unità di disegno G2D.

Funzionalità supportate:

Diverse funzionalità di disegno in LVGL possono essere demandate al motore G2D. La CPU è disponibile per altre operazioni mentre G2D è in esecuzione. È necessario il sistema operativo Linux per bloccare il thread di disegno LVGL e passare a un altro task o sospendere la CPU per risparmiare energia.

Le attività di disegno supportate sono disponibili in "src/draw/nx/g2d/lv_draw_g2d.c":

```
switch(t->type) {
    case LV_DRAW_TASK_TYPE_FILL:
        lv_draw_g2d_fill(u, t->draw_dsc, &t->area);
        break;
    case LV_DRAW_TASK_TYPE_IMAGE:
        lv_draw_g2d_img(u, t->draw_dsc, &t->area);
        break;
    default:
        break;
}
```

- Riempire l'area con un colore (senza raggio, senza gradiente) + opacità opzionale.
- Blit [Block Image Transfer] dell'immagine sorgente ARGB8888 sulla destinazione. ARGB8888 + opacità opzionale.
- Ridimensionare l'immagine sorgente ARGB8888.

Limitazioni note:

- G2D/PXP può ruotare solo ad angoli di 90x.
- La rotazione non è supportata per le immagini non allineate a blocchi di 16x16 pixel. G2D/PXP è impostato per elaborare blocchi 16x16 per ottimizzare il sistema in termini di larghezza di banda di memoria e tempo di elaborazione delle immagini. Il motore di output essenzialmente tronca tutti i pixel di output dopo che è stato scritto il numero desiderato di pixel. Quando si ruota un'immagine sorgente e l'output non è divisibile per la dimensione del blocco, i pixel errati potrebbero essere troncati e l'immagine di output finale potrebbe apparire spostata.
- La ricolorazione o la trasformazione di immagini con opacità o canale alfa non possono essere ottenute in una singola configurazione della pipeline G2D/PXP. Sarebbero necessari due o più passaggi.
- L'indirizzo del buffer deve essere allineato a 64 byte: impostare *`LV_DRAW_BUF_ALIGN`* a `64` in "lv_conf.h". Non è richiesto alcun allineamento di passo: impostare *`LV_DRAW_BUF_STRIDE_ALIGN`* a `1` in "lv_conf.h".

Setup del progetto:

- Aggiungere i file sorgente relativi a G2D (e i relativi header, se disponibili) al progetto:
  - "src/draw/nxp/g2d/lv_draw_buf_g2d.c": callback del buffer di disegno
  - "src/draw/nxp/g2d/lv_draw_g2d_fill.c": riempimento dell'area
  - "src/draw/nxp/g2d/lv_draw_g2d_img.c": immagine blit (con ricolorazione o trasformazione opzionale)
  - "src/draw/nxp/g2d/lv_draw_g2d.c": inizializzazione dell'unità di disegno
  - "src/draw/nxp/g2d/lv_draw_g2d_buf_map.c": mappa hash per i buffer g2d
  - "src/draw/nxp/g2d/lv_g2d_utils.c": helper delle funzioni

4.4.5 Renesas

Driver Nativi

Renesas è un partner ufficiale di LVGL. Pertanto, LVGL include il supporto nativo per Dave2D (la GPU di Renesas) e gestiamo anche progetti Renesas pronti all'uso.

Dave2D

Dave2D è in grado di accelerare la maggior parte delle operazioni di disegno di LVGL:

- Disegno di rettangoli, anche con gradienti
- Disegno, ridimensionamento e rotazione di immagini
- Disegno di lettere
- Disegno di triangoli
- Disegno di linee

Poiché Dave2D lavora in background, la CPU è libera per altre attività. In pratica, durante il rendering, Dave2D può ridurre l'utilizzo della CPU dalla metà a un terzo, a seconda dell'applicazione.

GLCDC

GLCDC è una periferica di uscita grafica multistadio disponibile in diversi MCU Renesas. È in grado di pilotare pannelli LCD tramite un'interfaccia RGB altamente configurabile.

Maggiori informazioni sono disponibili nella *pagina dei driver*.

Driver MPU

Le MPU Renesas (ovvero la famiglia RZ/G) possono utilizzare questi driver MPU LVGL.

Wayland

L'utilizzo del driver Wayland è supportato sulle schede madri con un desktop Wayland, ovvero un compositore Wayland.

fbdev

Il driver video LVGL `fbdev` (frame buffer) può essere utilizzato quasi sempre.

RA Family

Schede supportate nella RA Family:

- **EK-RA8D1**
- **EK-RA8P1**
- **EK-RA6M3G**

Esecuzione del Progetto

- L'IDE ufficiale di Renesas si chiama e^2 studio. Essendo basato su Eclipse, funziona anche su Windows, Linux e Mac. I componenti della famiglia RA richiedono determinate versioni minime del Flexible Software Package (FSP). Per informazioni, consultare i singoli file README.md nei repository applicabili. Gli FSP possono essere scaricati qui. Si noti che a partire dalla versione 6.0 di FSP, LVGL fa ora parte della sezione middleware dei componenti. Pertanto, quando si clonano le ultime versioni dei repository delle schede LVGL supportate, non è più necessario l'aggiornamento ricorsivo dei sottomoduli git.
- JLink viene utilizzato per il debug e può essere scaricato qui.
- Clonare il repository pronto all'uso per la scheda selezionata, per versioni FSP precedenti alla 6.0:

  ```
  git clone https://github.com/lvgl/lv_port_renesas_ek-ra8d1.git␣
  ↪--recurse-submodules
  ```

  Scaricare il file `.zip` da GitHub non funziona perché non scarica i sottomoduli.
- Clonare il repository pronto all'uso per la scheda selezionata, per versioni FSP dalla 6.0 in poi:

  ```
  git clone https://github.com/lvgl/lv_port_renesas_ek-ra8d1.git
  ```

- Aprire e^2 studio, andare su `File` -> `Import project` e selezionare `General/Existing projects into workspace`.
- Selezionare la cartella clonata e premere `Finish`.
- Doppio-click su `configuration.xml`. Questo attiverà la `Configuration Window`.

  Il Flexible Software Package (FSP) di Renesas include il supporto per i livelli BSP e HAL, esteso a diverse varianti di RTOS e altri stack middleware. I componenti saranno disponibili tramite la generazione di codice, incluso il punto di ingresso in *"main.c"*.

  Premere `Generate Project Content` nell'angolo in alto a destra.

- Compilare il progetto premendo `Ctrl` + `Alt` + `B`
- Cliccare sul pulsante Debug ( ). Se viene richiesto `Debug Configurations`, sul tab `Debugger` selezionare il `J-Link ARM` come `Debug hardware` e l'IC appropriato come `Target Device`:

- `R7FA8D1BH` per EK-RA8D1

- `R7FA6M3AH` per EK-RA6M3G

Nota

Sulle schede EK-RA8D1, il DIP switch 7 `SW1` (al centro della board) deve essere impostato su ON, tutti gli altri su OFF.

Si noti inoltre che l'apertura di un progetto precedentemente creato su FSP prima della versione 6.0 aprirà una finestra di dialogo che chiederà all'utente se desidera migrare al nuovo FSP v6.0. La migrazione non interromperà il progetto.

Modificare il progetto

Aprire una demo

Il punto di ingresso del task principale è contenuto in `src/LVGL_thread_entry.c`.

È possibile disabilitare le demo LVGL (o semplicemente commentarle) e chiamare alcune funzioni `lv_example_...()`, oppure aggiungere codice personalizzato.

Configurazione

`src/lv_conf.h` (o `src/lv_conf_user.h`) contiene le impostazioni più importanti per LVGL. Ovvero:

- *`LV_COLOR_DEPTH`* per impostare la profondità di colore predefinita di LVGL
- *`LV_MEM_SIZE`* per impostare la RAM massima disponibile per LVGL
- `LV_USE_DAVE2D` per abilitare la GPU

I componenti hardware e software possono essere modificati visivamente utilizzando la `Configuration Window`.

A partire da e2studio 2025-04.1 e FSP 6.0.0, LVGL è un componente gestito.

Supporto

In caso di problemi o domande, aprire una "issue" nel repository corrispondente.

RX Family

Schede supportate nella RX Family:

- **RX72N Envision Kit**

Esecuzione del Progetto

- L'IDE ufficiale di Renesas si chiama e^2 studio. Essendo basato su Eclipse, funziona anche su Windows, Linux e Mac. Può essere scaricato qui.
- Scaricare e installare il driver necessario per il debugger
  - per Windows: 64 bit qui e 32 bit qui
  - per Linux: qui
- RX72 richiede un compilatore esterno per il core RXv3. Una versione gratuita e open source è disponibile qui dopo la registrazione.

  Il compilatore deve essere attivato in e^2 studio:
  - Si va su `Help` -> `Add Renesas Toolchains`
  - Si preme il pulsante `Add...`
  - Si seleziona la cartella di installazione della toolchain

- Clonare il repository già pronto lv_port_renesas_rx72n-envision-kit:

```
git clone https://github.com/lvgl/lv_port_renesas_rx72n-envision-kit.
↪git --recurse-submodules
```

  Scaricare il file `.zip` da GitHub non funziona perché non scarica i sottomoduli.
- Aprire e^2 studio, andare su `File` -> `Import project` e selezionare `General/Existing projects into workspace`
- Selezionare la cartella clonata e premere `Finish`.
- Doppio-click su `RX72N_EnVision_LVGL.scfg` per attivare la `Configuration Window`.

  Lo Smart Configurator (SMC) di Renesas include il supporto per i layer BSP e HAL esteso a più varianti RTOS e altri stack middleware. I componenti saranno disponibili tramite la generazione di codice, incluso il punto di ingresso dell'applicazione.

Premere `Generate Code` nell'angolo in alto a destra.

- Compilare il progetto premendo `Ctrl` + `Alt` + `B`
- Cliccare sul pulsante Debug ( ). Se viene richiesto `Debug Configurations`, sul tab `Debugger` selezionare il `E2 Lite` come `Debug hardware` e `R5F572NN` come `Target Device`:

> **Nota**
>
> Assicurarsi che entrambi i canali del DIP switch `SW1` (accanto a `ECN1`) siano OFF.

Modificare il progetto

Aprire una demo

Il punto di ingresso del task principale è contenuto in `src/LVGL_thread_entry.c`.

È possibile disabilitare le demo LVGL (o semplicemente commentarle) e chiamare alcune funzioni `lv_example_...()`, oppure aggiungere codice personalizzato.

Configurazione

`src/lv_conf.h` contiene le impostazioni più importanti per LVGL. Ovvero:

- *`LV_COLOR_DEPTH`* per impostare la profondità di colore predefinita di LVGL
- *`LV_MEM_SIZE`* per impostare la RAM massima disponibile per LVGL

- `LV_USE_DAVE2D` per abilitare la GPU

I componenti hardware e software possono essere modificati visivamente utilizzando la `Configuration Window`.

Supporto

In caso di problemi o domande, aprire una "issue" nel repository corrispondente.

RZ/G Family

Schede supportate nella RZ/G Family:

- **RZ/G2L-EVKIT**
- **RZ/G2UL-EVKIT**

Esecuzione del Progetto

- Le schede RZ/G sono MPU con supporto per Linux. I progetti vengono sviluppati utilizzando un SDK con toolchain CLI cross-compilatore in un ambiente PC Linux (WSL, Docker, ecc. possono essere utilizzati su Windows).
- La G2L ha un desktop Wayland e il progetto appare come una finestra Wayland. La G2UL non ha un desktop, quindi il progetto è a schermo intero e utilizza il driver `fbdev`.
- L'SDK attualmente utilizza LVGL v8.3, quindi questo progetto utilizza questa versione per rispecchiare la versione dell'SDK, anche se LVGL è linkato staticamente. È possibile provare a utilizzare versioni più recenti di LVGL. Consultare la v8-to-v9 Migration Guide per gli aspetti da considerare.
- Clonare il repository pronto all'uso per la board selezionata:

  ```
  git clone https://github.com/lvgl/lv_port_renesas_rz-g2l-evkit␣
  ↪--recurse-submodules
  ```

  Scaricare il file `.zip` da GitHub non funziona perché non scarica i sottomoduli.
- Seguire le istruzioni nel file README.md del progetto per ottenere l'immagine della scheda SD e il programma di installazione della toolchain, compila e carica il progetto sulla scheda.
- Interrompere eventuali demo avviate automaticamente (su G2UL eseguire `systemctl stop demo-launcher` nel terminale).
- Esecuzione del progetto

  ```
  ./lvgl_demo_benchmark
  ```

Modificare il progetto

Aprire una demo

Il punto di ingresso è contenuto in `src/main.c`.

È possibile disabilitare le demo LVGL (`lv_demo_benchmark()`) (o semplicemente commentarle) e chiamare alcune funzioni `lv_example_...()`, oppure aggiungere codice personalizzato.

Configurazione

Modificare `lv_conf.h` e `lv_drv_conf.h` per configurare LVGL. L'immagine della board contiene LVGL e lv_drivers come librerie linkabili dinamicamente. Questo progetto compila LVGL staticamente per la personalizzazione e per il porting del benchmark LVGL v9 da LVGL v8.3.

Supporto

In caso di problemi o domande, aprire una "issue" nel repository corrispondente.

RZ/A Family

Schede supportate nella RZ/A Family:

- **RZ/A3M**

Esecuzione del Progetto

- Le schede RZ/A sono MPU con un focus su applicazioni bare metal e RTOS. I progetti vengono creati utilizzando l'IDE e2 Studio, disponibile per Windows, Mac e Linux.
- Clonare il repository pronto all'uso per la propria board:

  ```
  git clone https://github.com/lvgl/lv_port_renesas-ek-rz_a3m --
  ↪recurse-submodules
  ```

  Scaricare il file `.zip` da GitHub non funziona perché non scarica i sottomoduli.
- Seguire le istruzioni nel file README.md del progetto per compilare e flashare il progetto sulla board.

Modificare il progetto

Aprire una demo

Il punto di ingresso è contenuto in `src/LVGL_thread_entry.c`.

È possibile disabilitare le demo LVGL (`lv_demo_benchmark()`) (o semplicemente commentarle) e chiamare alcune funzioni `lv_example_...()`, oppure aggiungere codice personalizzato.

Configurazione

Modificare `lv_conf.h` per configurare LVGL.

Eseguirà automaticamente qualsiasi demo abilitata in `lv_conf.h`. È possibile consultare `lv_conf.defaults` per un riepilogo delle configurazioni modificate rispetto a quelle di default.

Supporto

In caso di problemi o domande, aprire una "issue" nel repository corrispondente.

Board Supportate

Nome della Board	CPU	Memoria	Display	Board video	Link
EK-RA8D1	480 MHz Arm Cortex-M85	1MB SDRAM interna 64MB SDRAM esterna 2MB flash interna 64MB octo-SPI flash esterna	4.5” 480x854 2-lane MIPI		Repository delle demo
EK-RA8P1	1 GHz Arm Cortex-M85	1872KB interna 128MB esterna 1MB flash interna	7” 1024x600 Parallel RGB888		Repository demo
EK-RA6M3G	120 MHz Arm Cortex-M4	640kB SRAM interna 2MB SRAM interna 32MB QSPI flash esterna	4.3” 480x272 Parallel RGB565		Repository demo
RX72N Envision Kit	240 MHz Renesas RXv3	1MB SRAM interna 4MB SRAM interna 32MB QSPI flash esterna	4.3” 480x272 Parallel RGB565		Repository demo
RZ/G2L-EVKIT	1.2 GHz Arm Cortex-A55	2GB DDR4 SDRAM	1920x1080 Micro-HDMI		Repository demo
RZ/G2UL-EVK	1.0 GHz Arm Cortex-A55	1GB DDR4 SDRAM	1280x800 Parallel RGB888 con Micro-HDMI		Repository demo
RZ/A3M	1.0 GHz Arm Cortex-A55	128MB DDR3L-SDRAM interna 128KB SRAM interna 128MB QSPI NAND flash esterna 32MB QSPI	4.3” 1280x720 MIPI		Repository demo

4.4.6 STM32

LVGL può essere aggiunto a STM32CubeIDE in modo simile a qualsiasi altro IDE basato su Eclipse.

Inclusione di LVGL in un Progetto

- Creare o aprire un progetto in STM32CubeIDE.
- Copiare l'intera cartella LVGL in *[project_folder]/Drivers/lvgl*.
- Nel riquadro **Project Explorer** di STM32CubeIDE: cliccare col pulsante destro del mouse sulla cartella LVGL copiata (potrebbe essere necessario aggiornare la vista prima che venga visualizzata) e selezionare **Add/remove include path…**. Se questa opzione non viene visualizzata o non funziona, si può controllare i path di inclusione del progetto nel menù **Project** -> **Properties**, quindi andare a **C/C++ Build** -> **Settings** -> **Include paths** e assicurarti che la directory LVGL sia elencata.

Ora che i file sorgente sono inclusi nel progetto, seguire le istruzioni per *aggiungere LVGL al progetto* e creare il file `lv_conf.h`, quindi inizializzare il display.

Esempio Bare Metal

Un esempio minimale che utilizza STM32CubeIDE e HAL. * Quando si imposta **Pinout and Configuration** utilizzando il **Device Configuration Tool**, selezionare **System Core** -> **SYS** e assicurarsi che **Timebase Source** sia impostato su **SysTick**. * Configurare eventuali altre periferiche (incluso il pannello LCD) e inizializzarle in *main.c*. * `#include "lvgl.h"` nel file *main.c*. * Creare alcuni frame buffer come variabili globali:

```
/* Frame buffers
 * Static or global buffer(s). The second buffer is optional
 * TODO: Adjust color format and choose buffer size. DISPLAY_WIDTH * 10 is␣
↪one suggestion. */
#define BYTES_PER_PIXEL (LV_COLOR_FORMAT_GET_SIZE(LV_COLOR_FORMAT_RGB565))␣
↪/* will be 2 for RGB565 */
#define BUFF_SIZE (DISPLAY_WIDTH * 10 * BYTES_PER_PIXEL)
static uint8_t buf_1[BUFF_SIZE];
static uint8_t buf_2[BUFF_SIZE];
```

- **Nella funzione `main()`, dopo aver inizializzato la CPU,**
  le periferiche e il pannello LCD, chiamare *`lv_init()`* per inizializzare LVGL. Si può poi creare il driver video usando *`lv_display_create()`* e registrare i frame buffer usando *`lv_display_set_buffers()`*.

```
//Initialise LVGL UI library
lv_init();

lv_display_t * disp = lv_display_create(WIDTH, HEIGHT); /* Basic␣
↪initialization with horizontal and vertical resolution in pixels␣
↪*/
lv_display_set_flush_cb(disp, my_flush_cb); /* Set a flush␣
↪callback to draw to the display */
lv_display_set_buffers(disp, buf_1, buf_2, sizeof(buf_1), LV_
↪DISPLAY_RENDER_MODE_PARTIAL); /* Set an initialized buffer */
```

- Creare alcuni widget fittizi per testare l'output:

```
/* Change Active Screen's background color */
lv_obj_set_style_bg_color(lv_screen_active(), lv_color_
↪hex(0x003a57), LV_PART_MAIN);
lv_obj_set_style_text_color(lv_screen_active(), lv_color_
↪hex(0xffffff), LV_PART_MAIN);
```

(continues on next page)

(continua dalla pagina precedente)

```
/* Create a spinner */
lv_obj_t * spinner = lv_spinner_create(lv_screen_active());
lv_obj_set_size(spinner, 64, 64);
lv_obj_align(spinner, LV_ALIGN_BOTTOM_MID, 0, 0);
```

- Aggiungere una chiamata a *lv_timer_handler()* all'interno del ciclo `while(1)`:

```
/* Infinite loop */
while (1)
{
    lv_timer_handler();
    HAL_Delay(5);
}
```

- Aggiungere una chiamata a *lv_tick_inc()* all'interno della funzione `SysTick_Handler()`. Aprire il file *stm32xxxx_it.c* (il nome dipenderà dalla propria MCU specifica) e aggiornare la funzione `SysTick_Handler()`:

```
void SysTick_Handler(void)
{
    /* USER CODE BEGIN SysTick_IRQn 0 */

    HAL_SYSTICK_IRQHandler();
    lv_tick_inc(1);
    #ifdef USE_RTOS_SYSTICK
        osSystickHandler();
    #endif

    /* USER CODE END SysTick_IRQn 0 */
    HAL_IncTick();
    /* USER CODE BEGIN SysTick_IRQn 1 */

    /* USER CODE END SysTick_IRQn 1 */
}
```

- Infine, scrivere la funzione di callback, `my_flush_cb`, che invierà il buffer del display al pannello LCD. Di seguito è riportato un esempio, ma varierà a seconda della configurazione.

```
void my_flush_cb(lv_display_t * disp, const lv_area_t * area, lv_color_
↪t * color_p)
{
    //Set the drawing region
    set_draw_window(area->x1, area->y1, area->x2, area->y2);

    int height = area->y2 - area->y1 + 1;
    int width = area->x2 - area->x1 + 1;

    //We will do the SPI write manually here for speed
    HAL_GPIO_WritePin(DC_PORT, DC_PIN, GPIO_PIN_SET);
    //CS low to begin data
    HAL_GPIO_WritePin(CS_PORT, CS_PIN, GPIO_PIN_RESET);

    //Write colour to each pixel
    for (int i = 0; i < width * height; i++) {
        uint16_t color_full = (color_p->red << 11) | (color_p->green <
↪< 5) | (color_p->blue);
        parallel_write(color_full);
```

(continues on next page)

(continua dalla pagina precedente)

```
        color_p++;
    }

    //Return CS to high
    HAL_GPIO_WritePin(CS_PORT, CS_PIN, GPIO_PIN_SET);

    /* IMPORTANT!!!
     * Inform the graphics library that you are ready with the␣
↪flushing */
    lv_display_flush_ready(disp);
}
```

Esempio FreeRTOS

Un esempio minimo che utilizza STM32CubeIDE, HAL e CMSISv1 (FreeRTOS). *Notare che non abbiamo utilizzato i mutex in questo esempio, tuttavia LVGL * **NON* è "thread safe" e quindi è consigliabile utilizzare i mutex.* Vedere: *Considerazioni sui Thread* * `#include "lvgl.h"` * Create your frame buffer(s) as global variables:

```
/* Frame buffers
 * Static or global buffer(s). The second buffer is optional */
#define BYTES_PER_PIXEL (LV_COLOR_FORMAT_GET_SIZE(LV_COLOR_FORMAT_RGB565))␣
↪/* will be 2 for RGB565 */
/* TODO: Declare your own BUFF_SIZE appropriate to your system. */
#define BUFF_SIZE (DISPLAY_WIDTH * 10 * BYTES_PER_PIXEL)
static uint8_t buf_1[BUFF_SIZE];
static uint8_t buf_2[BUFF_SIZE];
```

- Nella funzione `main`, dopo aver inizializzato le periferiche (SPI, GPIO, LCD ecc.), inizializzare LVGL usando *`lv_init()`*, creare un nuovo driver video usando *`lv_display_create()`* e registrare i frame buffer usando *`lv_display_set_buffers()`*.

```
/* Initialise LVGL UI library */
lv_init();
lv_display_t *display = lv_display_create(WIDTH, HEIGHT); /* Create␣
↪the display */
lv_display_set_flush_cb(display, my_flush_cb);            /* Set a␣
↪flush callback to draw to the display */
lv_display_set_buffers(display, buf_1, buf_2, sizeof(buf_1), LV_
↪DISPLAY_RENDER_MODE_PARTIAL); /* Set an initialized buffer */

/* Register the touch controller with LVGL - Not included here for␣
↪brevity. */
```

- Creare alcuni widget fittizi per testare l'output:

```
/* Change Active Screen's background color */
lv_obj_set_style_bg_color(lv_screen_active(), lv_color_hex(0x003a57),␣
↪LV_PART_MAIN);
lv_obj_set_style_text_color(lv_screen_active(), lv_color_hex(0xffffff),
↪ LV_PART_MAIN);

/* Create a spinner */
lv_obj_t * spinner = lv_spinner_create(lv_screen_active());
lv_obj_set_size(spinner, 64, 64);
lv_obj_align(spinner, LV_ALIGN_BOTTOM_MID, 0, 0);
```

- Creare due thread per chiamare *lv_timer_handler()* e *lv_tick_inc()*. Ci sarà bisogno di due handle `osThreadId` per CMSISv1. In questo caso, non è necessario che siano accessibili globalmente, tuttavia la generazione del codice STM32Cube lo fa di default. Se si usa CMSIS e la generazione del codice STM32Cube, il risultato dovrebbe essere simile a questo:

```
//Thread Handles
osThreadId lvgl_tickHandle;
osThreadId lvgl_timerHandle;

/* definition and creation of lvgl_tick */
osThreadDef(lvgl_tick, LVGLTick, osPriorityNormal, 0, 1024);
lvgl_tickHandle = osThreadCreate(osThread(lvgl_tick), NULL);

//LVGL update timer
osThreadDef(lvgl_timer, LVGLTimer, osPriorityNormal, 0, 1024);
lvgl_timerHandle = osThreadCreate(osThread(lvgl_timer), NULL);
```

- E creare le funzioni del thread:

```
/* LVGL timer for tasks. */
void LVGLTimer(void const * argument)
{
  for(;;)
  {
    lv_timer_handler();
    osDelay(20);
  }
}
/* LVGL tick source */
void LVGLTick(void const * argument)
{
  for(;;)
  {
    lv_tick_inc(10);
    osDelay(10);
  }
}
```

- Infine, creare la funzione `my_flush_cb` per inviare il frame buffer all'LCD. Le specifiche di questa funzione variano a seconda delle funzionalità della MCU in uso. Di seguito è riportato un semplice esempio di interfaccia LCD parallela; adattarla in base alle specifiche capacità del proprio display e della MCU.

```
void my_flush_cb(lv_display_t * display, const lv_area_t * area, uint8_
↪t * px_map)
{
  uint16_t * color_p = (uint16_t *)px_map;

  //Set the drawing region
  set_draw_window(area->x1, area->y1, area->x2, area->y2);

  int height = area->y2 - area->y1 + 1;
  int width = area->x2 - area->x1 + 1;

  //Begin SPI Write for DATA
  HAL_GPIO_WritePin(DC_PORT, DC_PIN, GPIO_PIN_SET);
  HAL_GPIO_WritePin(CS_PORT, CS_PIN, GPIO_PIN_RESET);

  //Write colour to each pixel
```

(continues on next page)

(continua dalla pagina precedente)

```
  for (int i = 0; i < width * height; i++) {
      parallel_write(color_p);
      color_p++;
  }

  //Return CS to high
  HAL_GPIO_WritePin(CS_PORT, CS_PIN, GPIO_PIN_SET);

  /* IMPORTANT!!!
   * Inform the graphics library that you are ready with the flushing␣
↪*/
  lv_display_flush_ready(display);
}
```

Supporto DMA2D

LVGL supporta DMA2D, una funzionalità di alcuni MCU STM32 che può migliorare le prestazioni durante i riempimenti "blending" e immagini. Alcune linee di prodotti STM32, come STM32F4, STM32F7, STM32L4, STM32U5 e STM32H7, includono modelli con supporto DMA2D.

L'integrazione di LVGL con DMA2D può essere abilitata impostando `LV_USE_DRAW_DMA2D` a `1` in `lv_conf.h`

Con `LV_USE_DRAW_DMA2D_INTERRUPT` impostato a `0` e `LV_USE_OS` impostato a `LV_OS_NONE`, DMA2D disegnerà alcuni riempimenti e immagini contemporaneamente al rendering software, ove possibile. Se `LV_USE_DRAW_DMA2D_INTERRUPT` è impostato a `1` e `LV_USE_OS` impostato a `LV_OS_FREERTOS` (o un altro sistema operativo), la differenza principale sarà che il core rimarrà inattivo invece di "busywait" in attesa del completamento di un trasferimento DMA2D.

Se `LV_USE_DRAW_DMA2D_INTERRUPT` è abilitato, è necessario chiamare *`lv_draw_dma2d_transfer_complete_interrupt_handler`* ogni volta che viene ricevuto l'interrupt globale DMA2D "transfer complete".

DMA2D consente inoltre di combinare layer con formato colore su `LV_COLOR_FORMAT_ARGB1555` sopra i layer `LV_COLOR_FORMAT_RGB565`.

Se il dispositivo STM ha una GPU NeoChrom, si può usare invece il *Nema GFX renderer*.

4.5 Driver

4.5.1 Display

Driver Framebuffer Linux

Panoramica

Il framebuffer Linux (fbdev) è un sottosistema Linux utilizzato per visualizzare la grafica. Si tratta di un'API indipendente dall'hardware che consente al software nello spazio utente di accedere al framebuffer (la parte della memoria video di un computer contenente un frame video corrente) utilizzando solo le funzionalità di base del kernel Linux e la sua interfaccia del file system del dispositivo, evitando la necessità di librerie che implementino i driver video nello spazio utente.

Prerequisiti

Il sistema dispone di un dispositivo framebuffer configurato (solitamente in `/dev/fb0`).

Configurazione del driver

Abilitare il supporto del driver framebuffer in lv_conf.h, tramite la definizione del compilatore cmake o tramite KConfig. È inoltre possibile configurare la modalità di rendering.

```
#define LV_USE_LINUX_FBDEV           1
#define LV_LINUX_FBDEV_RENDER_MODE   LV_DISPLAY_RENDER_MODE_PARTIAL
```

Utilizzo

Per impostare un display basato su framebuffer, creare prima un display con `lv_linux_fbdev_create`. Successivamente, si imposta il nodo del dispositivo framebuffer sul display (di solito è `/dev/fb0`).

```
lv_display_t *disp = lv_linux_fbdev_create();
lv_linux_fbdev_set_file(disp, "/dev/fb0");
```

Se lo schermo rimane nero o disegna solo parzialmente, si può provare ad abilitare il rendering diretto tramite `LV_DISPLAY_RENDER_MODE_DIRECT`. Inoltre, si può attivare una modalità di aggiornamento forzato con `lv_linux_fbdev_set_force_refresh(true)`. Questo di solito ha un impatto sulle prestazioni e non dovrebbe essere abilitato a meno che non sia realmente necessario.

Nascondere il cursore

Si potrebbe vedere un cursore lampeggiante sullo schermo. Il metodo per nasconderlo varia a seconda della piattaforma. Ad esempio, ecco come si può fare su un Raspberry Pi:

1. Modificare il file `/boot/cmdline.txt`.
2. Aggiungere `vt.global_cursor_default=0`.

Errori comuni

Problema della risoluzione di default

Quando il kernel Linux si inizializza, imposta sottosistemi come il framebuffer (fbdev) per gestire l'output del display. Se è collegato un display HDMI, il kernel lo rileva e alloca una porzione di RAM come framebuffer, che contiene i dati pixel per il rendering delle immagini.

Tuttavia, un problema comune si verifica quando il display non viene acceso durante il processo di avvio. Se la scheda madre viene accesa prima dello schermo, il kernel potrebbe configurare erroneamente la risoluzione del framebuffer. Di conseguenza, sia la risoluzione visibile che quella virtuale possono essere errate, causando problemi di visualizzazione.

Questo problema si verifica spesso con le connessioni HDMI in cui il display viene acceso dopo che il sistema si è già avviato.

Il seguente comando restituisce le impostazioni correnti del dispositivo framebuffer specificato, come risoluzione, profondità di pixel e timing.

```
fbset -fb /dev/fb0
```

Per evitare problemi relativi al display, si consiglia di assicurarsi che tutti i dispositivi, incluso il display HDMI, siano collegati e accesi prima di accendere la scheda madre.

FT81x

Un driver framebuffer minimale per i controller smart display EVE FT81x. Funziona anche per BT81x.

Al momento è supportata la modalità di rendering parziale a buffer singolo Al momento potrebbe non supportare DSPI o QSPI.

Testato a 32 MHz su FT812 e a 23 MHz su BT817.

Utilizzo

```
#define FB_SIZE             800 * 2 * 50
#define MAX_TRANSFER_SIZE   FB_SIZE

static void spi_cb(lv_display_t * disp, lv_ft81x_spi_operation operation,␣
↪void * data, uint32_t length)
{
    spi_device_handle_t spi = lv_ft81x_get_user_data(disp);
    switch(operation) {
        case LV_FT81X_SPI_OPERATION_CS_ASSERT:
            gpio_set_level(CS_PIN, 0);
            break;
        case LV_FT81X_SPI_OPERATION_CS_DEASSERT:
            gpio_set_level(CS_PIN, 1);
            esp_rom_delay_us(10); /* tiny delay in case a CS_ASSERT␣
↪immediately follows */
            break;
        case LV_FT81X_SPI_OPERATION_SEND: {
            spi_transaction_t trans = {0};
            while(length) {
                uint32_t sz = length < MAX_TRANSFER_SIZE ? length : MAX_
↪TRANSFER_SIZE;
                trans.length = sz * 8;
                trans.rxlength = 0;
                trans.tx_buffer = data;
                spi_device_polling_transmit(spi, &trans);
                length -= sz;
                data += sz;
            }
            break;
        }
        case LV_FT81X_SPI_OPERATION_RECEIVE: {
            spi_transaction_t trans = {0};
            trans.length = length * 8;
            trans.rxlength = length * 8;
            trans.rx_buffer = data;
            spi_device_polling_transmit(spi, &trans);
            break;
        }
    }
}

int main() {
    /* ... */

    // reset the ft81x
    gpio_set_level(PD_PIN, 0);
    vTaskDelay(6 / portTICK_PERIOD_MS);
    gpio_set_level(PD_PIN, 1);
    vTaskDelay(21 / portTICK_PERIOD_MS);

    /* ... */

    // taken from https://github.com/lvgl/lvgl_esp32_drivers/blob/
↪9fed1cc47b5a45fec6bae08b55d2147d3b50260c/lvgl_tft/EVE_config.h
    // NHD-5.0-800480FT-CxXx-xxx 800x480 5.0" Newhaven, resistive or␣
```

(continues on next page)

(continua dalla pagina precedente)

```
↪capacitive, FT81x
    // EVE_NHD_50
    #define EVE_VSYNC0      (0L)
    #define EVE_VSYNC1      (3L)
    #define EVE_VOFFSET     (32L)
    #define EVE_VCYCLE      (525L)
    #define EVE_HSYNC0      (0L)
    #define EVE_HSYNC1      (48L)
    #define EVE_HOFFSET     (88L)
    #define EVE_HCYCLE      (928L)
    #define EVE_PCLKPOL     (0L)
    #define EVE_SWIZZLE     (0L)
    #define EVE_PCLK        (2L)
    #define EVE_CSPREAD     (1L)

    lv_ft81x_parameters_t params = {
        .hor_res = 800,
        .ver_res = 480,

        .hcycle = EVE_HCYCLE,
        .hoffset = EVE_HOFFSET,
        .hsync0 = EVE_HSYNC0,
        .hsync1 = EVE_HSYNC1,
        .vcycle = EVE_VCYCLE,
        .voffset = EVE_VOFFSET,
        .vsync0 = EVE_VSYNC0,
        .vsync1 = EVE_VSYNC1,
        .swizzle = EVE_SWIZZLE,
        .pclkpol = EVE_PCLKPOL,
        .cspread = EVE_CSPREAD,
        .pclk = EVE_PCLK,

        .has_crystal = true,
        .is_bt81x = false
    };

    static uint8_t fb[FB_SIZE] __attribute__((aligned(4)));
    lv_display_t * disp = lv_ft81x_create(&params, fb, FB_SIZE, spi_cb,␣
↪spi);

    /* ... */
}
```

Risoluzione dei problemi

Se il "backlight " [retroilluminazione] non si accende (o è troppo luminoso o troppo fioco), provare a modificare il valore di `PWM_DUTY_BACKLIGHT_ON` in `lv_ft81x.c`, che può variare a seconda della board.

Driver generico per controller LCD compatibile con MIPI DCS

Panoramica

Da Wikipedia:

> MIPI Alliance è un'alleanza commerciale globale che sviluppa specifiche tecniche per l'ecosistema mobile, in particolare per gli smartphone, ma include anche i settori influenzati dalla tecnologia mobile. MIPI è stata fondata nel 2003 da Arm, Intel, Nokia, Samsung, STMicroelectronics e Texas Instruments.

MIPI Alliance ha pubblicato una serie di specifiche relative ai dispositivi di visualizzazione, tra cui DBI (Display Bus Interface), DSI (Display Serial Interface) e DCS (Display Command Set). Di solito, quando si parla di un display compatibile con MIPI, si pensa a un dispositivo con interfaccia seriale DSI. Tuttavia, la specifica Display Bus Interface include una serie di altre interfacce legacy, come l'interfaccia seriale SPI o l'interfaccia parallela compatibile con i8080, che vengono spesso utilizzate per interfacciare i display LCD con microcontrollori di fascia bassa. Inoltre, la specifica DCS contiene un set di comandi standard, supportato da un gran numero di controller LCD TFT legacy, inclusi i popolari SOC Sitronix (ST7735, ST7789, ST7796) e Ilitek (ILI9341). Questi comandi forniscono un'interfaccia comune per configurare l'orientamento del display, la risoluzione del colore, varie modalità di alimentazione e forniscono l'accesso generico alla memoria video. Oltre a questo set di comandi standard, ogni chip controller LCD dispone di una serie di comandi specifici del fornitore per configurare i livelli del generatore di tensione, le temporizzazioni o le curve gamma.

Nota

È importante comprendere che questo driver LCD MIPI generico non è un driver hardware per display con interfaccia seriale DSI ("MIPI"). Invece, implementa il set di comandi MIPI DCS utilizzato in molti controller LCD con bus SPI o i8080 e fornisce un framework comune per i controller display specifici per chip.

Suggerimento

Sebbene si tratti di un driver generico, può essere utilizzato per supportare chip compatibili che non dispongono di un driver specifico.

Prerequisiti

Non ci sono prerequisiti.

Configurazione del driver

Abilitare il supporto del driver LCD MIPI generico in lv_conf.h, tramite la definizione del compilatore cmake o tramite KConfig.

```
#define LV_USE_GENERIC_MIPI  1
```

Nota

LV_USE_GENERIC_MIPI viene abilitato automaticamente quando viene abilitato un driver compatibile.

Utilizzo

È necessario implementare due funzioni dipendenti dalla piattaforma:

```
/* Send short command to the LCD. This function shall wait until the␣
↪transaction finishes. */
int32_t my_lcd_send_cmd(lv_display_t *disp, const uint8_t *cmd, size_t cmd_
↪size, const uint8_t *param, size_t param_size)
{
        ...
}

/* Send large array of pixel data to the LCD. If necessary, this function␣
↪has to do the byte-swapping. This function can do the transfer in the␣
↪background. */
```

(continues on next page)

(continua dalla pagina precedente)

```
int32_t my_lcd_send_color(lv_display_t *disp, const uint8_t *cmd, size_t␣
↪cmd_size, uint8_t *param, size_t param_size)
{
        ...
}
```

L'unica differenza tra le funzioni `my_lcd_send_cmd()` e `my_lcd_send_color()` è che `my_lcd_send_cmd()` viene utilizzata per inviare comandi brevi e ci si aspetta che completi la transazione al suo ritorno (in altre parole, dovrebbe essere bloccante), mentre `my_lcd_send_color()` viene utilizzata solo per inviare dati pixel ed è consigliabile utilizzare DMA per trasmettere i dati in background. È possibile implementare anche metodi più sofisticati, come l'accodamento dei trasferimenti e la loro pianificazione in background.

Si noti che, sebbene il flushing del display sia gestito dal driver, è responsabilità dell'utente chiamare *`lv_display_flush_ready()`* al termine del trasferimento del colore. In caso di trasferimento DMA, questa operazione viene solitamente eseguita in una callback di "transfer ready".

Nota

Sebbene sia accettabile utilizzare un'implementazione bloccante anche per il trasferimento dei pixel, le prestazioni ne risentiranno.

Suggerimento

È necessario prestare attenzione a non inviare un comando mentre è in corso un trasferimento attivo in background. È responsabilità dell'utente implementare questa funzionalità interrogando l'hardware, interrogando una variabile globale (che viene reimpostata al termine del trasferimento) o utilizzando un semaforo o un altro meccanismo di blocco.

Si noti inoltre che il driver non gestisce l'allocazione del "draw buffer" [buffer di disegno], poiché anche questa potrebbe dipendere dalla piattaforma. Pertanto, è necessario allocare i buffer e assegnarli all'oggetto display come di consueto chiamando *`lv_display_set_buffers()`*.

Il driver può essere utilizzato per creare più display. In una configurazione di questo tipo, le callback devono essere in grado di distinguere tra i display. In genere si implementa un set separato di callback per ciascun display. Si noti inoltre che l'utente deve occuparsi dell'arbitraggio del bus quando più dispositivi sono collegati ad esso.

Esempio

Nota

È possibile trovare una guida passo passo e l'implementazione effettiva dei callback su un STM32F746 utilizzando STM32CubeIDE e le librerie ST HAL qui: *Guida passo passo: come utilizzare i driver LCD LVGL v9 con i dispositivi STM32*

```
#include "src/drivers/display/st7789/lv_st7789.h"

#define LCD_H_RES               240
#define LCD_V_RES               320
#define LCD_BUF_LINES   60

lv_display_t *my_disp;
```

(continues on next page)

(continua dalla pagina precedente)

```
...

/* Initialize LCD I/O bus, reset LCD */
static int32_t my_lcd_io_init(void)
{
        ...
        return HAL_OK;
}

/* Send command to the LCD controller */
static void my_lcd_send_cmd(lv_display_t *disp, const uint8_t *cmd, size_t␣
↪cmd_size, const uint8_t *param, size_t param_size)
{
        ...
}

/* Send pixel data to the LCD controller */
static void my_lcd_send_color(lv_display_t *disp, const uint8_t *cmd, size_
↪t cmd_size, uint8_t *param, size_t param_size)
{
        ...

        /* This must be called to signal that the transfer has finished.
         * It is typically called in a "DMA transfer complete" callback
         * long after `my_lcd_send_color` has returned.
         */
        lv_display_flush_ready(disp);
}

int main(int argc, char ** argv)
{
        ...

        /* Initialize LVGL */
        lv_init();

        /* Initialize LCD bus I/O */
        if (my_lcd_io_init() != 0)
                return;

        /* Create the LVGL display object and the LCD display driver */
        my_disp = lv_lcd_generic_mipi_create(LCD_H_RES, LCD_V_RES, LV_LCD_
↪FLAG_NONE, my_lcd_send_cmd, my_lcd_send_color);

        /* Set display orientation to landscape */
        lv_display_set_rotation(my_disp, LV_DISPLAY_ROTATION_90);

        /* Configure draw buffers, etc. */
        uint8_t * buf1 = NULL;
        uint8_t * buf2 = NULL;

        uint32_t buf_size = LCD_H_RES * LCD_BUF_LINES * lv_color_format_
↪get_size(lv_display_get_color_format(my_disp));

        buf1 = lv_malloc(buf_size);
        if(buf1 == NULL) {
```

(continues on next page)

(continua dalla pagina precedente)

```
                LV_LOG_ERROR("display draw buffer malloc failed");
                return;
        }
        /* Allocate secondary buffer if needed */
        ...

        lv_display_set_buffers(my_disp, buf1, buf2, buf_size, LV_DISPLAY_
↪RENDER_MODE_PARTIAL);

        ui_init(my_disp);

        while(true) {
                ...

                /* Periodically call the lv_timer handler */
                lv_timer_handler();
        }
}
```

Argomenti avanzati

Creazione di flag

Il terzo argomento della funzione *lv_lcd_generic_mipi_create()* è un array di flag. Questo può essere utilizzato per configurare l'orientamento e l'ordinamento RGB del pannello se le impostazioni di default non funzionano. In particolare, il driver MIPI generico accetta i seguenti flag:

```
LV_LCD_FLAG_NONE
LV_LCD_FLAG_MIRROR_X
LV_LCD_FLAG_MIRROR_Y
LV_LCD_FLAG_BGR
```

È possibile passare più flag combinandoli tramite OR, ad esempio, *LV_LCD_FLAG_MIRROR_X | LV_LCD_FLAG_BGR*.

Elenchi di comandi personalizzati

Sebbene i driver specifici per chip facciano del loro meglio per inizializzare correttamente il controller LCD, è possibile che diversi pannelli TFT richiedano configurazioni diverse. In particolare, una corretta impostazione della gamma è fondamentale per una buona riproduzione dei colori. Purtroppo, trovare un buon set di parametri non è facile. Di solito il produttore del pannello fornisce un codice di esempio con le impostazioni di registro consigliate.

È possibile utilizzare la funzione `my_lcd_send_cmd()` per inviare un comando arbitrario al controller LCD. Tuttavia, per semplificare l'invio di un numero elevato di parametri, il driver MIPI generico supporta l'invio di un elenco di comandi personalizzati al controller. I comandi devono essere inseriti in un array di 'uint8_t':

```
static const uint8_t init_cmd_list[] = {
        <command 1>, <number of parameters>, <parameter 1>, ... <parameter␣
↪N>,
        <command 2>, <number of parameters>, <parameter 1>, ... <parameter␣
↪N>,
        ...
        LV_LCD_CMD_DELAY_MS, LV_LCD_CMD_EOF             /* terminate list:␣
↪this is required! */
};

...
```

(continues on next page)

(continua dalla pagina precedente)

```
lv_lcd_generic_mipi_send_cmd_list(my_disp, init_cmd_list);
```

È possibile aggiungere un ritardo tra i comandi utilizzando lo pseudo-comando `LV_LCD_CMD_DELAY_MS`, seguito dal ritardo specificato in unità di 10 ms. Per terminare l'elenco dei comandi, è necessario utilizzare un ritardo con valore `LV_LCD_CMD_EOF`, come mostrato sopra.

Vedere un esempio concreto di invio di un elenco di comandi qui.

Driver per controller LCD ILI9341

Panoramica

ILI9341 è un driver SOC a chip singolo da 262.144 colori per display a cristalli liquidi a-TFT con risoluzione di 240 RGB x 320 punti, composto da un driver sorgente a 720 canali, un driver di gate a 320 canali, 172.800 byte di GRAM per dati di visualizzazione grafica di 240 RGB x 320 punti e un circuito di alimentazione. ILI9341 supporta l'interfaccia MCU parallela con bus dati a 8/9/16/18 bit, l'interfaccia RGB con bus dati a 6/16/18 bit e l'interfaccia periferica seriale (SPI) a 3/4 linee.

Il driver del controller LCD ILI9341 è un driver indipendente dalla piattaforma, basato sul drive generico MIPI. Implementa l'inizializzazione del display, supporta la rotazione del display e implementa il callback di flush del display. L'utente deve implementare solo due funzioni specifiche della piattaforma per inviare un comando o dati pixel al controller tramite SPI o bus parallelo. In genere, queste vengono implementate richiamando le funzioni della libreria SDK appropriate sulla piattaforma specificata.

Prerequisiti

Non ci sono prerequisiti.

Configurazione del driver

Abilitare il supporto del driver ILI9341 in lv_conf.h, tramite la definizione del compilatore cmake o tramite KConfig.

```
#define LV_USE_ILI9341  1
```

Utilizzo

È necessario implementare due funzioni dipendenti dalla piattaforma:

```
/* Send short command to the LCD. This function shall wait until the␣
↪transaction finishes. */
int32_t my_lcd_send_cmd(lv_display_t *disp, const uint8_t *cmd, size_t cmd_
↪size, const uint8_t *param, size_t param_size)
{
        ...
}

/* Send large array of pixel data to the LCD. If necessary, this function␣
↪has to do the byte-swapping. This function can do the transfer in the␣
↪background. */
int32_t my_lcd_send_color(lv_display_t *disp, const uint8_t *cmd, size_t␣
↪cmd_size, uint8_t *param, size_t param_size)
{
        ...
}
```

Per creare un display basato su ILI9341, utilizzare la funzione

```
/**
 * Create an LCD display with ILI9341 driver
 * @param hor_res       horizontal resolution
 * @param ver_res       vertical resolution
 * @param flags         default configuration settings (mirror, RGB␣
↪ordering, etc.)
 * @param send_cmd      platform-dependent function to send a command to␣
↪the LCD controller (usually uses polling transfer)
 * @param send_color    platform-dependent function to send pixel data to␣
↪the LCD controller (usually uses DMA transfer: must implement a 'ready'␣
↪callback)
 * @return              pointer to the created display
 */
lv_display_t * lv_ili9341_create(uint32_t hor_res, uint32_t ver_res, lv_
↪lcd_flag_t flags,
                                                                lv_ili9341_
↪send_cmd_cb_t send_cmd_cb, lv_ili9341_send_color_cb_t send_color_cb);
```

Per ulteriori dettagli e un esempio pratico, consultare la documentazione generica del driver MIPI.

> **Nota**
>
> È possibile trovare una guida passo passo e l'implementazione effettiva dei callback su un STM32F746 utilizzando STM32CubeIDE e le librerie ST HAL qui: *Guida passo passo: come utilizzare i driver LCD LVGL v9 con i dispositivi STM32*

Guida passo passo: come utilizzare i driver LCD LVGL v9 con i dispositivi STM32

Introduzione

Questa guida intende fornire istruzioni dettagliate su come configurare l'HAL STM32Cube con i nuovi driver per display TFT-LCD introdotti in LVGL v9.0. Il codice di esempio è stato testato sulla scheda Nucleo-F746ZG basata su STM32F746 con un pannello LCD basato su ST7789 connesso tramite SPI. L'applicazione stessa e il codice di configurazione hardware sono stati generati con il tool STM32CubeIDE 1.14.0.

> **Suggerimento**
>
> ST Micro fornisce i propri driver TFT-LCD nel pacchetto di estensione software X-CUBE-DISPLAY. Sebbene questi driver possano essere utilizzati anche con LVGL, i driver LCD LVGL non dipendono da questo pacchetto.
>
> I driver LCD LVGL sono concepiti come un'API alternativa e semplice da usare per implementare il supporto LCD per i progetti basati su LVGL su qualsiasi piattaforma. Inoltre, anche nella versione iniziale supportiamo più controller LCD di quanti ne fornisca attualmente X-CUBE-DISPLAY e prevediamo di aggiungere il supporto per un numero ancora maggiore di controller LCD in futuro.
>
> Si noti tuttavia che, a differenza di X-CUBE-DISPLAY, i driver LCD LVGL non implementano la parte di comunicazione, che sia SPI, bus parallelo i8080 o altro. È responsabilità dell'utente implementarli e ottimizzarli sulla piattaforma scelta. LVGL fornirà solo esempi per le piattaforme più diffuse.

Seguendo i passaggi descritti, si otterrà un programma completamente funzionante, utilizzabile come base per un progetto basato su LVGL. Se si ha fretta e non si è interessati ai dettagli, si può trovare il progetto finale qui. Si dovrà solo configurare LVGL per utilizzare il driver corrispondente al proprio hardware (se diverso dall'ST7789) e implementare la funzione `ui_init()` per creare i propri widget.

Nota

Questo esempio non intende rappresentare la migliore implementazione possibile, né la soluzione consigliata. Si basa esclusivamente sui driver HAL forniti da ST Micro, che privilegiano la portabilità rispetto alle prestazioni. Ciò nonostante, le prestazioni sono molto buone, grazie all'efficiente implementazione dei driver basata su DMA.

Nota

Sebbene l'esempio utilizzi FreeRTOS, questo non è un requisito obbligatorio per i driver dei display LCD LVGL.

È possibile trovare frammenti di codice sorgente di questa guida nell'esempio lv_port_lcd_stm32_template.c.

Configurazione hardware

In questo esempio utilizzeremo la periferica SPI1 per collegare il microcontrollore al pannello LCD. Oltre ai pin SPI controllati dall'hardware SCK e MOSI, abbiamo bisogno di alcuni pin di uscita aggiuntivi per la selezione del chip, la selezione dei comandi/dati e il reset dell'LCD:

pin	configurazione	LCD	label utente
PA4	GPIO_Output	CS	LCD_CS
PA5	SPI1_SCK	SCK	--
PA7	SPI1_MOSI	SDI	--
PA15	GPIO_Output	RESET	LCD_RESET
PB10	GPIO_Output	DC	LCD_DCX

Istruzioni passo passo

1. Creare un nuovo progetto in File/New/STM32.
2. Selezionare il processore/scheda target.
3. Impostare il nome e la posizione del progetto.
4. Impostare il "Targeted Project Type" su STM32Cube e premere Finish.
5. Rispondi "Yes" per inizializzare le periferiche col loro "Mode" di default? Dopo la creazione del progetto, il file di configurazione (.ioc) si apre automaticamente.
6. Passare alla scheda "Pinout & Configuration".
7. Nella categoria "System Core" passare a RCC.
8. Impostare "High Speed Clock" a "BYPASS Clock Source" e "Low Speed Clock" a "Crystal/Ceramic Resonator".
9. Nella categoria "System Core" selezionare SYS e impostare "Timebase Source" su un valore diverso da SysTick (nel nostro esempio, TIM2).
10. Passare alla ""Clock Configuration".
11. Impostare la frequenza di clock HCLK al valore massimo (216 MHz per STM32F746).
12. Tornare alla scheda "Pinout & Configuration" e, nella categoria "Middleware and Software Packs", selezionare FREERTOS.
13. Selezionare "Interface: CMSIS_V1".
14. Nella scheda "Advanced Settings" abilitare USE_NEWLIB_REENTRANT. Abbiamo terminato.
15. Nella vista "Pinout" configurare PA5 su SPI1_SCK, PA7 su SPI1_MOSI (clic destro del mouse sul pin e si seleziona la funzione).

16. Nella categoria "Pinout & Configuration/Connectivity" selezionare SPI1.
17. Impostare "Mode" su "Transmit Only Master" e "Hardware NSS Signal" a Disable.
18. Nella sottofinestra "Configuration" passare a "Parameter Settings".
19. Impostare "Frame Format" su Motorola, "Data Size" su "8 Bits", "First Bit" su "MSB First".
20. Impostare il Prescaler al valore massimo in base alla scheda tecnica del controller LCD (ad esempio, 15 MBit/s). Impostare "CPOL/CPHA" come richiesto (lasciare i valori di default).
21. Impostare "NSSP Mode" su Disabled e "NSS Signal Type" su Software.
22. In "DMA Settings" aggiungere una nuova "Request" per SPI1_TX (quando si usa SPI1).
23. Impostare "Priority" su Medium, "Data Width" su "Half Word".
24. In "NVIC Settings" abilitare l'interrupt globale SPI1.
25. In "GPIO Settings" impostare "SPI1_SCK" su "Pull-down" e "Very High output speed" e impostare "User Label" su `LCD_SCK`.
26. impostare "SPI1_MOSI" su Pull-up e Very High, e rinominarlo `LCD_SDI`.
27. Selezionare la categoria "System Core/GPIO". Nella vista Pinout configurare pin aggiuntivi per la selezione del chip, il reset e la selezione dei comandi/dati. Chiamarli rispettivamente `LCD_CS`, `LCD_RESET` e `LCD_DCX`. Configurarli come "GPIO Output". (In questo esempio useremo PA4 per `LCD_CS`, PA15 per `LCD_RESET` e PB10 per `LCD_DCX`.)
28. Impostare `LCD_CS` su "No pull-up" e "no pull-down", "Low level" e "Very High speed".
29. Impostare `LCD_RESET` to "Pull-up" e "High level".
30. Impostare `LCD_DCX` su "No pull-up" e "no pull-down", "High level" e "Very High speed".
31. Aprire la scheda "Project Manager" e selezionare "Advanced Settings". Sul lato destro si trova la finestra "Register Callback". Selezionare SPI e impostarlo su ENABLE.
32. La configurazione hardware è pronta. Salvare la configurazione e lasciare che STM32Cube generi il codice sorgente.
33. Nell'alberatura del progetto, clonare il repository LVGL nella cartella "Middlewares/Third_Party" (questo tutorial utilizza il branch "release/v9.0" di LVGL):

```
git clone https://github.com/lvgl/lvgl.git -b release/v9.0
```

34. La clonazione dovrebbe creare una sottocartella 'lvgl' all'interno della cartella 'Third_Party'. Dalla cartella 'lvgl' copiare 'lv_conf_template.h' nella cartella 'Middlewares' e rinominarlo in 'lv_conf.h'. Aggiorna l'alberatura del progetto.
35. Aprire 'lv_conf.h' e alla riga 15 cambiare `#if 0` in `#if 1`.
36. Cercare la stringa `LV_USE_ST7735` e abilitare il driver LCD appropriato impostandone il valore su 1. Questo esempio utilizza il driver ST7789:

```
#define LV_USE_ST7789           1
```

37. Click destro sulla cartella 'Middlewares/Third_Party/lvgl/tests', selezionare "Resource Configurations/Exclude from Build...", selezionare sia Debug che Release, quindi premere OK.
38. Click destro sul nome del progetto e selezionare "Properties". Nel pannello "C/C++ Build/Settings" selezionare MCU GCC Compiler/Include paths. Nel menu a discesa Configuration selezionare "[All configurations]". Aggiungere il seguente "Include path":

```
../Middlewares/Third_Party/lvgl
```

39. Aprire Core/Src/stm32xxx_it.c (il nome del file dipende dalla variante del processore). Aggiungere 'lv_tick.h' alla sezione "Private includes":

```
/* Private includes ----------------------------------------
↪---------------*/
/* USER CODE BEGIN Includes */
#include "./src/tick/lv_tick.h"
/* USER CODE END Includes */
```

40. Trovare la funzione `TIM2_IRQHandler`. Aggiungere una chiamata a `lv_tick_inc()`:

```
void TIM2_IRQHandler(void)
{
  /* USER CODE BEGIN TIM2_IRQn 0 */

  /* USER CODE END TIM2_IRQn 0 */
  HAL_TIM_IRQHandler(&htim2);
  /* USER CODE BEGIN TIM2_IRQn 1 */
  lv_tick_inc(1);
  /* USER CODE END TIM2_IRQn 1 */
}
```

41. Salvare il file, quindi aprire "Core/Src/main.c". Aggiungere le seguenti righe a "Private includes" (se il vostro LCD utilizza un driver diverso da ST7789, sostituite il path e l'header del driver con quelli appropriati):

```
/* Private includes ----------------------------------------
↪---------------*/
/* USER CODE BEGIN Includes */
#include "lvgl.h"
#include "./src/drivers/display/st7789/lv_st7789.h"
/* USER CODE END Includes */
```

42. Aggiungere le seguenti righe alle definizioni private (modificarle in base alle specifiche del vostro LCD):

```
#define LCD_H_RES       240
#define LCD_V_RES       320
#define BUS_SPI1_POLL_TIMEOUT 0x1000U
```

43. Aggiungere le seguenti righe a "Private variables":

```
osThreadId LvglTaskHandle;
lv_display_t *lcd_disp;
volatile int lcd_bus_busy = 0;
```

44. Aggiungere la seguente riga a "Private function prototypes":

```
void ui_init(lv_display_t *disp);
void LVGL_Task(void const *argument);
```

45. Aggiungere le seguenti righe dopo USER CODE BEGIN RTOS_THREADS:

```
osThreadDef(LvglTask, LVGL_Task, osPriorityIdle, 0, 1024);
LvglTaskHandle = osThreadCreate(osThread(LvglTask), NULL);
```

46. Copiare e incollare le funzioni di inizializzazione hardware e di callback di trasferimento dal codice di esempio dopo USER CODE BEGIN 4:

```
/* USER CODE BEGIN 4 */

void lcd_color_transfer_ready_cb(SPI_HandleTypeDef *hspi)
{
```

(continues on next page)

(continua dalla pagina precedente)

```
        /* CS high */
        HAL_GPIO_WritePin(LCD_CS_GPIO_Port, LCD_CS_Pin, GPIO_
↪PIN_SET);
        lcd_bus_busy = 0;
        lv_display_flush_ready(lcd_disp);
}

/* Initialize LCD I/O bus, reset LCD */
static int32_t lcd_io_init(void)
{
        /* Register SPI Tx Complete Callback */
        HAL_SPI_RegisterCallback(&hspi1, HAL_SPI_TX_COMPLETE_
↪CB_ID, lcd_color_transfer_ready_cb);

        /* reset LCD */
        HAL_GPIO_WritePin(LCD_RESET_GPIO_Port, LCD_RESET_Pin,␣
↪GPIO_PIN_RESET);
        HAL_Delay(100);
        HAL_GPIO_WritePin(LCD_RESET_GPIO_Port, LCD_RESET_Pin,␣
↪GPIO_PIN_SET);
        HAL_Delay(100);

        HAL_GPIO_WritePin(LCD_CS_GPIO_Port, LCD_CS_Pin, GPIO_
↪PIN_SET);
        HAL_GPIO_WritePin(LCD_DCX_GPIO_Port, LCD_DCX_Pin, GPIO_
↪PIN_SET);

        return HAL_OK;
}

/* Platform-specific implementation of the LCD send command␣
↪function. In general this should use polling transfer. */
static void lcd_send_cmd(lv_display_t *disp, const uint8_t␣
↪*cmd, size_t cmd_size, const uint8_t *param, size_t param_
↪size)
{
        LV_UNUSED(disp);
        while (lcd_bus_busy);   /* wait until previous␣
↪transfer is finished */
        /* Set the SPI in 8-bit mode */
        hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
        HAL_SPI_Init(&hspi1);
        /* DCX low (command) */
        HAL_GPIO_WritePin(LCD_DCX_GPIO_Port, LCD_DCX_Pin, GPIO_
↪PIN_RESET);
        /* CS low */
        HAL_GPIO_WritePin(LCD_CS_GPIO_Port, LCD_CS_Pin, GPIO_
↪PIN_RESET);
        /* send command */
        if (HAL_SPI_Transmit(&hspi1, cmd, cmd_size, BUS_SPI1_
↪POLL_TIMEOUT) == HAL_OK) {
                /* DCX high (data) */
                HAL_GPIO_WritePin(LCD_DCX_GPIO_Port, LCD_DCX_
↪Pin, GPIO_PIN_SET);
                /* for short data blocks we use polling␣
↪transfer */
```

(continues on next page)

(continua dalla pagina precedente)

```
                HAL_SPI_Transmit(&hspi1, (uint8_t *)param,␣
↪(uint16_t)param_size, BUS_SPI1_POLL_TIMEOUT);
                /* CS high */
                HAL_GPIO_WritePin(LCD_CS_GPIO_Port, LCD_CS_Pin,
↪ GPIO_PIN_SET);
        }
}

/* Platform-specific implementation of the LCD send color␣
↪function. For better performance this should use DMA␣
↪transfer.
 * In case of a DMA transfer a callback must be installed to␣
↪notify LVGL about the end of the transfer.
 */
static void lcd_send_color(lv_display_t *disp, const uint8_t␣
↪*cmd, size_t cmd_size, uint8_t *param, size_t param_size)
{
        LV_UNUSED(disp);
        while (lcd_bus_busy);   /* wait until previous␣
↪transfer is finished */
        /* Set the SPI in 8-bit mode */
        hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
        HAL_SPI_Init(&hspi1);
        /* DCX low (command) */
        HAL_GPIO_WritePin(LCD_DCX_GPIO_Port, LCD_DCX_Pin, GPIO_
↪PIN_RESET);
        /* CS low */
        HAL_GPIO_WritePin(LCD_CS_GPIO_Port, LCD_CS_Pin, GPIO_
↪PIN_RESET);
        /* send command */
        if (HAL_SPI_Transmit(&hspi1, cmd, cmd_size, BUS_SPI1_
↪POLL_TIMEOUT) == HAL_OK) {
                /* DCX high (data) */
                HAL_GPIO_WritePin(LCD_DCX_GPIO_Port, LCD_DCX_
↪Pin, GPIO_PIN_SET);
                /* for color data use DMA transfer */
                /* Set the SPI in 16-bit mode to match␣
↪endianness */
                hspi1.Init.DataSize = SPI_DATASIZE_16BIT;
                HAL_SPI_Init(&hspi1);
                lcd_bus_busy = 1;
                HAL_SPI_Transmit_DMA(&hspi1, param, (uint16_
↪t)param_size / 2);
                /* NOTE: CS will be reset in the transfer␣
↪ready callback */
        }
}
```

47. Aggiungere la funzione LVGL_Task(). Sostituire la chiamata `lv_st7789_create()` con il driver appropriato. È possibile modificare l'orientamento di default modificando il parametro di `lv_display_set_rotation()`. Sarà inoltre necessario creare i "display buffer". Questo esempio utilizza uno schema di doppio buffering con buffer parziali di dimensioni pari a 1/10. Nella maggior parte dei casi questo è un buon compromesso tra la dimensione della memoria richiesta e le prestazioni, ma si può sperimentare con altre impostazioni.

```
void LVGL_Task(void const *argument)
```

(continues on next page)

(continua dalla pagina precedente)

```
{
        /* Initialize LVGL */
        lv_init();

        /* Initialize LCD I/O */
        if (lcd_io_init() != 0)
                return;

        /* Create the LVGL display object and the LCD display␣
↪driver */
        lcd_disp = lv_st7789_create(LCD_H_RES, LCD_V_RES, LV_
↪LCD_FLAG_NONE, lcd_send_cmd, lcd_send_color);
        lv_display_set_rotation(lcd_disp, LV_DISPLAY_ROTATION_
↪270);

        /* Allocate draw buffers on the heap. In this example␣
↪we use two partial buffers of 1/10th size of the screen */
        lv_color_t * buf1 = NULL;
        lv_color_t * buf2 = NULL;

        uint32_t buf_size = LCD_H_RES * LCD_V_RES / 10 * lv_
↪color_format_get_size(lv_display_get_color_format(lcd_disp));

        buf1 = lv_malloc(buf_size);
        if(buf1 == NULL) {
                LV_LOG_ERROR("display draw buffer malloc failed
↪");
                return;
        }

        buf2 = lv_malloc(buf_size);
        if(buf2 == NULL) {
                LV_LOG_ERROR("display buffer malloc failed");
                lv_free(buf1);
                return;
        }
        lv_display_set_buffers(lcd_disp, buf1, buf2, buf_size,␣
↪LV_DISPLAY_RENDER_MODE_PARTIAL);

        ui_init(lcd_disp);

        for(;;) {
                /* The task running lv_timer_handler should␣
↪have lower priority than that running `lv_tick_inc` */
                lv_timer_handler();
                /* raise the task priority of LVGL and/or␣
↪reduce the handler period can improve the performance */
                osDelay(10);
        }
}
```

48. Non resta che implementare `ui_init()` per creare lo screen. Ecco un semplice esempio di "Hello World":

```
    void ui_init(lv_display_t *disp)
    {
            lv_obj_t *obj;
```

(continues on next page)

(continua dalla pagina precedente)

```
            /* set screen background to white */
            lv_obj_t *scr = lv_screen_active();
            lv_obj_set_style_bg_color(scr, lv_color_white(),␣
↪0);
            lv_obj_set_style_bg_opa(scr, LV_OPA_100, 0);

            /* create label */
            obj = lv_label_create(scr);
lv_obj_set_align(widget, LV_ALIGN_CENTER);
lv_obj_set_height(widget, LV_SIZE_CONTENT);
lv_obj_set_width(widget, LV_SIZE_CONTENT);
lv_obj_set_style_text_font(widget, &lv_font_montserrat_14, 0);
lv_obj_set_style_text_color(widget, lv_color_black(), 0);
lv_label_set_text(widget, "Hello World!");
    }
```

NXP eLCDIF

Panoramica

eLCDIF è una periferica presente su alcuni dispositivi NXP in grado di pilotare pannelli di display tramite l'interfaccia RGB, supporta diverse profondità di colore e, sui dispositivi compatibili con MIPI-DSI, il suo output può essere indirizzato all'interfaccia fisica del display MIPI. Il driver NXP eLCDIF di LVGL è responsabile del "bind" [collegamento] del driver di basso livello dell'SDK MCUx di NXP al sottosistema di visualizzazione LVGL.

Prerequisiti

- Questo driver si basa sulla presenza dell'SDK MCUx di NXP nello stesso progetto.
- Attivare il driver impostando `LV_USE_NXP_ELCDIF` a `1` nel file *"lv_conf.h"*.

Utilizzo

Il driver LVGL per eLCDIF presuppone che la piattaforma abbia già configurato il driver di basso livello del display, impostato il pin-mux, i clock, ecc. Richiede inoltre che l'indirizzo base della periferica e la struttura di configurazione siano già impostati.

Il codice seguente illustra l'utilizzo del diver in modalità `LV_DISPLAY_RENDER_MODE_DIRECT`. Si noti che in questa modalità di funzionamento l'applicazione è responsabile dell'allocazione dello spazio dei framebuffer e del loro passaggio al display. Nell'esempio seguente `buffer1` e `buffer2` sono il buffer corrente e quello successivo che verranno copiati sullo schermo, scambiati a ogni operazione di flush (gestita internamente dal driver del display). Si noti inoltre che, in "direct mode", ogni buffer dovrebbe avere spazio sufficiente per contenere almeno le dimensioni dello schermo, ovvero l'altezza per la larghezza per i byte di un pixel (dipendente dall'applicazione o supportato dal display). Nel codice seguente, questa dimensione è rappresentata da `buf_size`.

```
elcdif_rgb_mode_config_t config;
ELCDIF_RgbModeGetDefaultConfig(&config);

lv_display_t * g_disp = lv_nxp_display_elcdif_create_direct(LCDIF, config,␣
↪buffer1, buffer2, buf_size);
lv_display_set_default(g_disp);
```

Per utilizzare il driver in modalità `LV_DISPLAY_RENDER_MODE_PARTIAL`, è necessario allocare un buffer aggiuntivo, preferibilmente nella regione di memoria più veloce disponibile.

Lo swapping del buffer può essere attivato passando un secondo buffer della stessa dimensione al posto dell'argomento `NULL`. Si noti che in questo caso `BUF_SIZE` deve avere almeno uno spazio sufficiente a contenere dati pari a 1/10 delle dimensioni effettive del display.

```
#define BUF_SIZE (DISPLAY_HEIGHT * DISPLAY_WIDTH / 10 * 2) /*1/10 screen␣
↪size for RGB565 format*/
static uint8_t partial_draw_buf[BUF_SIZE];
lv_display_t * g_disp = lv_nxp_display_elcdif_create_partial(LCDIF, config,
↪ partial_draw_buf, NULL, BUF_SIZE);
```

In fase di runtime, la funzione di gestione eventi del driver eLCDIF deve essere chiamata all'interno del gestore di interrupt eLCDIF. Questa funzione è responsabile della notifica al sottosistema di visualizzazione LVGL del completamento dell'operazione di flush:

```
void eLCDIF_IRQ_Handler(void)
{
    lv_nxp_display_elcdif_event_handler(g_disp);
}
```

Renesas GLCDC

Panoramica

GLCDC è una periferica di output grafico multistadio utilizzata nei microcontrollori Renesas. È progettata per generare automaticamente segnali di temporizzazione e dati per diversi pannelli LCD.

- Supporta pannelli LCD con interfaccia RGB (fino a 24 bit) e segnali di sincronizzazione (HSYNC, VSYNC e Data Enable opzionali)
- Supporta vari formati colore per i piani grafici in ingresso (RGB888, ARGB8888, RGB565, ARGB1555, ARGB4444, CLUT8, CLUT4, CLUT1)
- Supporta l'utilizzo della Color Look-Up Table (CLUT) per i piani grafici in ingresso (ARGB8888) con 512 word (32 bit/word)
- Supporta vari formati colore per l'output (RGB888, RGB666, RGB565, Serial RGB888)
- Può inserire due piani grafici sopra il piano di background e fonderli sullo schermo
- Genera un dot clock per il pannello. La sorgente di clock è selezionabile tra interna o esterna (LCD_EXTCLK).
- Supporta la regolazione della luminosità, del contrasto e la correzione gamma.
- Supporta gli interrupt GLCDC per gestire la commutazione del frame buffer o il rilevamento dell'underflow.

La configurazione di un progetto e l'ulteriore integrazione con l'ecosistema Renesas sono descritte in dettaglio nella *pagina Renesas*. Consultare i seguenti repository per esempi pronti all'uso:

- EK-RA8D1
- EK-RA6M3G
- RX72N Envision Kit

Prerequisiti

- Questo diver si basa sul codice generato da e^2 studio. Omettere questo passaggio durante la configurazione del progetto causerà un errore di compilazione.
- Attivare il diver impostando `LV_USE_RENESAS_GLCDC` a `1` nel file *"lv_conf.h"*.

Utilizzo

Non è necessario implementare funzioni specifiche per la piattaforma.

Il codice seguente illustra l'utilizzo del diver in modalità `LV_DISPLAY_RENDER_MODE_DIRECT`.

```
lv_display_t * disp = lv_renesas_glcdc_direct_create();
lv_display_set_default(disp);
```

Per utilizzare il driver in modalità `LV_DISPLAY_RENDER_MODE_PARTIAL`, è necessario allocare un buffer aggiuntivo, preferibilmente nella regione di memoria più veloce disponibile.

Lo swapping del buffer può essere attivato passando un secondo buffer della stessa dimensione al posto dell'argomento `NULL`.

```
static lv_color_t partial_draw_buf[DISPLAY_HSIZE_INPUT0 * DISPLAY_VSIZE_
↪INPUT0 / 10] BSP_PLACE_IN_SECTION(".sdram") BSP_ALIGN_VARIABLE(1024);

lv_display_t * disp = lv_renesas_glcdc_partial_create(partial_draw_buf,␣
↪NULL, sizeof(partial_draw_buf));
lv_display_set_default(disp);
```

> **Nota**
>
> La modalità parziale può essere attivata tramite la macro nel file `src/board_init.c` dei progetti demo.

Rotazione dello schermo

La rotazione dello schermo basata su software è supportata in modalità parziale. Utilizza l'API comune, non è richiesta alcuna configurazione aggiuntiva:

```
lv_display_set_rotation(lv_display_get_default(), LV_DISP_ROTATION_90);
/* OR */
lv_display_set_rotation(lv_display_get_default(), LV_DISP_ROTATION_180);
/* OR */
lv_display_set_rotation(lv_display_get_default(), LV_DISP_ROTATION_270);
```

Assicurarsi che l'heap sia sufficientemente grande, poiché verrà allocato un buffer con le stesse dimensioni del buffer parziale.

Driver LTDC STM32

Alcuni STM32 dispongono di una periferica specializzata per la gestione dei display chiamata LTDC (LCD-TFT display controller).

Modalità di utilizzo con LVGL

Il driver all'interno di LVGL è progettato per funzionare con una periferica LTDC già configurata. Si basa sull'HAL per rilevare le informazioni sulla configurazione. Il formato colore del display LVGL creato corrisponderà al formato colore del layer LTDC. Utilizzare STM32CubeIDE o STM32CubeMX per generare il codice di inizializzazione LTDC.

Esistono diversi casi d'uso per il driver LVGL. Tutte le permutazioni delle opzioni seguenti sono ben supportate.

- buffer singolo o doppio
- modalità di rendering diretta o parziale
- Con sistema operativo e senza sistema operativo
- flushing parallelo con DMA2D (solo per la modalità di rendering parziale)

Se il sistema operativo è abilitato, verrà utilizzata una primitiva di sincronizzazione per dare al thread la possibilità di cedere il passo ad altri thread mentre è bloccato, migliorando l'utilizzo della CPU. Vedere *LV_USE_OS* in your lv_conf.h

Layer LTDC

Questo driver crea un display LVGL che si occupa solo di uno specifico layer della periferica LTDC, il che significa che è possibile creare due display LVGL LTDC che operano indipendentemente sui rispettivi layer.

Direct Render Mode

Per la modalità di rendering diretto, richiamare *lv_st_ltdc_create_direct()* in questo modo:

```
void * my_ltdc_framebuffer_address = (void *)0x20000000u;
uint32_t my_ltdc_layer_index = 0; /* typically 0 or 1 */
lv_display_t * disp = lv_st_ltdc_create_direct(my_ltdc_framebuffer_address,
                                               optional_other_full_size_
↪buffer,
                                               my_ltdc_layer_index);
```

`my_ltdc_framebuffer_address` è il framebuffer configurato per l'utilizzo da parte di LTDC. `optional_other_full_size_buffer` può essere un altro buffer delle stesse dimensioni del framebuffer di default per la modalità a doppio buffer, oppure `NULL` in caso contrario. `my_ltdc_layer_index` è l'indice del layer LTDC per cui creare il display.

Per ottenere i migliori risultati visivi, si consiglia di utilizzare `optional_other_full_size_buffer` se la memoria disponibile è sufficiente. La modalità a buffer singolo è quella da utilizzare se la memoria è molto scarsa. Se la memoria disponibile è quasi sufficiente per la modalità diretta a doppio buffer, ma non del tutto, utilizzare la modalità di rendering parziale.

Per chiarire cosa sia esattamente `my_ltdc_framebuffer_address`, si tratta del valore di `pLayerCfg.FBStartAdress` quando il layer LTDC viene configurato utilizzando l'HAL STM32, che viene scritto nel registro `CFBAR` della periferica del layer LTDC.

Partial Render Mode

Per la modalità di rendering parziale, invocare `lv_st_ltdc_create_partial()` in questo modo:

```
static uint8_t partial_buf1[65536];
static uint8_t optional_partial_buf2[65536];
uint32_t my_ltdc_layer_index = 0; /* typically 0 or 1 */
lv_display_t * disp = lv_st_ltdc_create_partial(partial_buf1,
                                                optional_partial_buf2,
                                                65536,
                                                my_ltdc_layer_index);
```

Il driver utilizzerà le informazioni nella configurazione del layer LTDC per trovare il framebuffer del layer e scaricarlo.

Fornire un secondo buffer parziale può migliorare l'utilizzo della CPU e aumentare le prestazioni rispetto a un singolo buffer se `LV_ST_LTDC_USE_DMA2D_FLUSH` è abilitato.

Script del Linker

È necessario assicurarsi che la memoria del framebuffer LTDC sia effettivamente riservata nello script del linker. Questo è un file che normalmente ha l'estensione `.ld`. Nell'esempio seguente, `1125K` è specificato perché la profondità di colore è 24 (3 byte per pixel), la larghezza del display è 800, l'altezza del display è 480 e `1K` significa 1024 byte. $3 \times 800 \times 480 \div 1024 = 1125$. La somma delle voci RAM (`FB_RAM` + `RAM`) dev'essere uguale alla RAM totale del dispositivo.

```
/* Memories definition */
MEMORY
{
    FB_RAM (xrw)   : ORIGIN = 0x20000000, LENGTH = 1125K /* single 24bit␣
↪800x480 buffer */
    RAM    (xrw)   : ORIGIN = 0x20119400, LENGTH = 1883K
    FLASH  (rx)    : ORIGIN = 0x08000000, LENGTH = 4096K
}
```

Rotazione del Display

Il driver supporta la rotazione del display con `lv_display_set_rotation(disp, rotation)` dove rotation è una tra `LV_DISP_ROTATION_90`, `LV_DISP_ROTATION_180` o `LV_DISP_ROTATION_270`. La rotazione è inizialmente `LV_DISP_ROTATION_0`.

La rotazione viene eseguita tramite software e funziona solo se il display è stato creato utilizzando `lv_st_ltdc_create_partial()`. `LV_ST_LTDC_USE_DMA2D_FLUSH` non avrà alcun effetto se si utilizza la rotazione.

DMA2D

`LV_ST_LTDC_USE_DMA2D_FLUSH` può essere abilitato per utilizzare DMA2D per svuotare i buffer parziali in parallelo con altre attività LVGL, indipendentemente dal fatto che il sistema operativo sia abilitato o meno. Se la visualizzazione non è parziale, non è necessario abilitare questa opzione.

Inoltre, è possibile combinare layer con formato colore `LV_COLOR_FORMAT_ARGB1555` sopra i layer `LV_COLOR_FORMAT_RGB565` utilizzando DMA2D.

Non deve essere abilitato contemporaneamente a `LV_USE_DRAW_DMA2D`. Vedere *DMA2D support*.

Further Reading

Si potrebbe essere interessati ad abilitare il *Nema GFX renderer* se l'STM32 ha una GPU NeoChrom.

lv_port_riverdi_stm32u5 è un modo rapido per iniziare a usare LTDC su LVGL.

Driver del controller LCD ST7735

Panoramica

ST7735S è un controller/driver a chip singolo per LCD TFT-LCD grafico a 262K colori. È costituito da 396 circuiti di pilotaggio per linee sorgente e 162 per linee di gate. Questo chip è in grado di connettersi direttamente a un microprocessore esterno e accetta interfacce SPI (Serial Peripheral Interface) e interfacce parallele a 8/9/16/18 bit. I dati del display possono essere memorizzati nella RAM integrata del display da 132 x 162 x 18 bit. Può eseguire operazioni di lettura/scrittura dei dati di visualizzazione nella RAM senza bisogno di un clock esterno, riducendo al minimo il consumo energetico. Inoltre, grazie ai circuiti di alimentazione integrati necessari per pilotare i cristalli liquidi, è possibile realizzare un sistema di visualizzazione con un numero inferiore di componenti.

Il driver del controller LCD ST7735, è un driver "platform-agnostic" [indipendente dalla piattaforma], basato sul driver generico MIPI. Implementa l'inizializzazione del display, supporta la rotazione del display e implementa il callback di flush del display. L'utente deve implementare solo due funzioni specifiche della piattaforma per inviare un comando o dati pixel al controller tramite SPI o bus parallelo. In genere, queste vengono implementate richiamando le funzioni della libreria SDK appropriate sulla piattaforma specificata.

Prerequisiti

Non ci sono prerequisiti.

Configurazione del driver

Abilitare il supporto del driver ST7735 in lv_conf.h, tramite la definizione del compilatore cmake o tramite KConfig

```
#define LV_USE_ST7735  1
```

Utilizzo

È necessario implementare due funzioni dipendenti dalla piattaforma:

```
/* Send short command to the LCD. This function shall wait until the␣
↪transaction finishes. */
int32_t my_lcd_send_cmd(lv_display_t *disp, const uint8_t *cmd, size_t cmd_
↪size, const uint8_t *param, size_t param_size)
{
        ...
}

/* Send large array of pixel data to the LCD. If necessary, this function␣
↪has to do the byte-swapping. This function can do the transfer in the␣
↪background. */
int32_t my_lcd_send_color(lv_display_t *disp, const uint8_t *cmd, size_t␣
↪cmd_size, uint8_t *param, size_t param_size)
{
        ...
}
```

Per creare un display basato su ST7735 utilizzare la funzione

```
/**
 * Create an LCD display with ST7735 driver
 * @param hor_res       horizontal resolution
 * @param ver_res       vertical resolution
 * @param flags         default configuration settings (mirror, RGB␣
↪ordering, etc.)
 * @param send_cmd      platform-dependent function to send a command to␣
↪the LCD controller (usually uses polling transfer)
 * @param send_color    platform-dependent function to send pixel data to␣
↪the LCD controller (usually uses DMA transfer: must implement a 'ready'␣
↪callback)
 * @return              pointer to the created display
 */
lv_display_t * lv_st7735_create(uint32_t hor_res, uint32_t ver_res, lv_lcd_
↪flag_t flags,
                                                                lv_st7735_
↪send_cmd_cb_t send_cmd_cb, lv_st7735_send_color_cb_t send_color_cb);
```

Per ulteriori dettagli e un esempio pratico, consultare la documentazione generica del driver MIPI.

> **Nota**
>
> È possibile trovare una guida passo passo e l'implementazione effettiva dei callback su un STM32F746 utilizzando STM32CubeIDE e le librerie ST HAL qui: *Guida passo passo: come utilizzare i driver LCD LVGL v9 con i dispositivi STM32*

Driver per Controller LCD ST7789

Panoramica

L'ST7789 è un controller/driver a chip singolo per TFT-LCD grafico a 262K colori. È costituito da 720 circuiti di pilotaggio per linee "source" e 320 per linee di gate. Questo chip è in grado di connettersi direttamente a un microprocessore esterno e accetta interfacce parallele a 8/9/16/18 bit. I dati del display possono essere memorizzati nella RAM integrata da 240x320x18 bit. Può eseguire operazioni di lettura/scrittura dei dati di visualizzazione nella RAM senza bisogno di un clock esterno, riducendo al minimo il consumo energetico. Inoltre, grazie al circuito di

alimentazione integrato necessario per pilotare i cristalli liquidi, è possibile realizzare un sistema di visualizzazione con il minor numero di componenti.

Il driver del controller LCD ST7789 è un driver indipendente dalla piattaforma, basato sul driver generico MIPI. Implementa l'inizializzazione del display, supporta la rotazione del display e implementa il callback di flush del display. L'utente deve implementare solo due funzioni specifiche della piattaforma per inviare un comando o dati pixel al controller tramite SPI o bus parallelo. In genere, queste vengono implementate richiamando le funzioni della libreria SDK appropriate sulla piattaforma specificata.

Prerequisiti

Non ci sono prerequisiti.

Configurazione del driver

Abilitare il supporto del driver ST7789 in lv_conf.h, tramite la definizione del compilatore cmake o tramite KConfig

```
#define LV_USE_ST7789  1
```

Utilizzo

È necessario implementare due funzioni dipendenti dalla piattaforma:

```
/* Send short command to the LCD. This function shall wait until the␣
↪transaction finishes. */
int32_t my_lcd_send_cmd(lv_display_t *disp, const uint8_t *cmd, size_t cmd_
↪size, const uint8_t *param, size_t param_size)
{
        ...
}

/* Send large array of pixel data to the LCD. If necessary, this function␣
↪has to do the byte-swapping. This function can do the transfer in the␣
↪background. */
int32_t my_lcd_send_color(lv_display_t *disp, const uint8_t *cmd, size_t␣
↪cmd_size, uint8_t *param, size_t param_size)
{
        ...
}
```

Per creare un display basato su ST7789 utilizzare la funzione

```
/**
 * Create an LCD display with ST7789 driver
 * @param hor_res       horizontal resolution
 * @param ver_res       vertical resolution
 * @param flags         default configuration settings (mirror, RGB␣
↪ordering, etc.)
 * @param send_cmd      platform-dependent function to send a command to␣
↪the LCD controller (usually uses polling transfer)
 * @param send_color    platform-dependent function to send pixel data to␣
↪the LCD controller (usually uses DMA transfer: must implement a 'ready'␣
↪callback)
 * @return              pointer to the created display
 */
lv_display_t * lv_st7789_create(uint32_t hor_res, uint32_t ver_res, lv_lcd_
↪flag_t flags,
```

(continues on next page)

(continua dalla pagina precedente)

```
                                                                    lv_st7789_
↪send_cmd_cb_t send_cmd_cb, lv_st7789_send_color_cb_t send_color_cb);
```

Per ulteriori dettagli e un esempio pratico, consultare la documentazione generica del driver MIPI.

> **Nota**
>
> È possibile trovare una guida passo passo e l'implementazione effettiva dei callback su un STM32F746 utilizzando STM32CubeIDE e le librerie ST HAL qui: *Guida passo passo: come utilizzare i driver LCD LVGL v9 con i dispositivi STM32*

Driver per Controller LCD ST7796

Panoramica

ST7796S è un controller/driver a chip singolo per display LCD TFT grafico a 262K colori. È costituito da 960 linee sorgente e 480 linee di gate per il pilotaggio dei circuiti. ST7796S è in grado di connettersi direttamente a un microprocessore esterno e accetta interfacce parallele a 8/9/16/18 bit, SPI; inoltre, fornisce anche un'interfaccia MIPI. I dati del display possono essere memorizzati nella RAM integrata da 320x480x18 bit. Può eseguire operazioni di lettura/scrittura della RAM dei dati del display senza clock esterno, riducendo al minimo il consumo energetico. Inoltre, grazie al circuito di alimentazione integrato necessario per pilotare i cristalli liquidi, è possibile realizzare un sistema di visualizzazione con un numero inferiore di componenti.

Il driver del controller LCD ST7796 è un driver indipendente dalla piattaforma, basato sul rive generico MIP. Implementa l'inizializzazione del display, supporta la rotazione del display e implementa il callback di flush del display. L'utente deve implementare solo due funzioni specifiche della piattaforma per inviare un comando o dati pixel al controller tramite SPI o bus parallelo. In genere, queste vengono implementate richiamando le funzioni della libreria SDK appropriate sulla piattaforma specificata.

Prerequisiti

Non ci sono prerequisiti.

Configurazione del driver

Abilitare il supporto del driver ST7796 in lv_conf.h, tramite la definizione del compilatore cmake o tramite KConfig

```
#define LV_USE_ST7796  1
```

Utilizzo

È necessario implementare due funzioni dipendenti dalla piattaforma:

```
/* Send short command to the LCD. This function shall wait until the␣
↪transaction finishes. */
int32_t my_lcd_send_cmd(lv_display_t *disp, const uint8_t *cmd, size_t cmd_
↪size, const uint8_t *param, size_t param_size)
{
        ...
}

/* Send large array of pixel data to the LCD. If necessary, this function␣
↪has to do the byte-swapping. This function can do the transfer in the␣
↪background. */
int32_t my_lcd_send_color(lv_display_t *disp, const uint8_t *cmd, size_t␣
```

(continues on next page)

(continua dalla pagina precedente)

```
↪cmd_size, uint8_t *param, size_t param_size)
{
        ...
}
```

Per creare un display basato su ST7796 utilizzare la funzione

```
/**
 * Create an LCD display with ST7796 driver
 * @param hor_res       horizontal resolution
 * @param ver_res       vertical resolution
 * @param flags         default configuration settings (mirror, RGB␣
↪ordering, etc.)
 * @param send_cmd      platform-dependent function to send a command to␣
↪the LCD controller (usually uses polling transfer)
 * @param send_color    platform-dependent function to send pixel data to␣
↪the LCD controller (usually uses DMA transfer: must implement a 'ready'␣
↪callback)
 * @return              pointer to the created display
 */
lv_display_t * lv_st7796_create(uint32_t hor_res, uint32_t ver_res, lv_lcd_
↪flag_t flags,
                                                                    lv_st7796_
↪send_cmd_cb_t send_cmd_cb, lv_st7796_send_color_cb_t send_color_cb);
```

Per ulteriori dettagli e un esempio pratico, consultare la documentazione generica del driver MIPI.

> **Nota**
>
> È possibile trovare una guida passo passo e l'implementazione effettiva dei callback su un STM32F746 utilizzando STM32CubeIDE e le librerie ST HAL qui: *Guida passo passo: come utilizzare i driver LCD LVGL v9 con i dispositivi STM32*

4.5.2 Driver Libinput

Panoramica

Libinput è uno stack di input per processi che devono fornire eventi da dispositivi di input di uso comune. Tra questi rientrano mouse, tastiere, touchpad, touchscreen e tavolette grafiche. Libinput gestisce le peculiarità specifiche del dispositivo e fornisce un'API di facile utilizzo per ricevere eventi dai dispositivi.

Prerequisiti

È installata la versione di sviluppo di libinput (di solito `libinput-dev`). Se il dispositivo di input richiede peculiarità, assicurarsi che siano installate (di solito in `/usr/share/libinput/*.quirks`). Per verificare se il dispositivo è configurato correttamente per l'utilizzo con libinput, è possibile eseguire `libinput list-devices`.

```
$ sudo libinput list-devices
...
Device:           ETPS/2 Elantech Touchpad
Kernel:           /dev/input/event5
Group:            10
Seat:             seat0, default
Size:             102x74mm
Capabilities:     pointer gesture
```

(continues on next page)

(continua dalla pagina precedente)

```
Tap-to-click:     disabled
Tap-and-drag:     enabled
...
```

Se il dispositivo non viene visualizzato, potrebbe essere necessario configurare udev e le regole udev appropriate per connetterlo.

Inoltre, per il supporto completo della tastiera, incluse lettere e modificatori, è necessario installare la versione di sviluppo di libxkbcommon (solitamente `libxkbcommon-dev`).

Configurazione del driver

Abilitare il supporto del driver libinput in lv_conf.h, tramite la definizione del compilatore cmake o tramite KConfig.

```
#define LV_USE_LIBINPUT    1
```

Il supporto completo della tastiera deve essere abilitato separatamente.

```
#define LV_LIBINPUT_XKB            1
#define LV_LIBINPUT_XKB_KEY_MAP    { .rules = NULL, .model = "pc101", .
↪layout = "us", .variant = NULL, .options = NULL }
```

Per trovare i valori corretti per la mappatura dei tasti, è possibile utilizzare il comando `setxkbmap -query`.

Utilizzo

Per impostare un dispositivo di input tramite il driver libinput, è sufficiente chiamare `lv_libinput_create` on il rispettivo tipo di dispositivo (`LV_INDEV_TYPE_POINTER` o `LV_INDEV_TYPE_KEYPAD`) e il path del nodo del dispositivo (ad esempio `/dev/input/event5`).

```
lv_indev_t *indev = lv_libinput_create(LV_INDEV_TYPE_POINTER, "/dev/input/
↪event5");
```

Si noti che i touchscreen vengono trattati come dispositivi di puntamento (assoluti) dal driver libinput e richiedono `LV_INDEV_TYPE_POINTER`.

A seconda del sistema, i path dei nodi dei dispositivi potrebbero non essere stabili dopo i riavvii. In questo caso, è possibile utilizzare `lv_libinput_find_dev` per trovare il primo dispositivo con una funzionalità specifica.

```
char *path = lv_libinput_find_dev(LV_LIBINPUT_CAPABILITY_TOUCH, true);
```

Il secondo argomento controlla se tutti i dispositivi vengono nuovamente scansionati o meno. Se si hanno molti dispositivi collegati, l'operazione può risultare piuttosto lenta. Pertanto, è consigliabile specificare `true` solo alla prima chiamata quando si richiama questo metodo più volte di seguito. Per trovare tutti i dispositivi con una funzionalità specifica, utilizzare `lv_libinput_find_devs`.

Per collegare una tastiera a un'area di testo, creare un gruppo di input dedicato e impostarlo sia sull'indev che sulla textarea.

```
lv_obj_t *textarea = lv_textarea_create(...);
...
lv_group_t *keyboard_input_group = lv_group_create();
lv_indev_set_group(indev, keyboard_input_group);
lv_group_add_obj(keyboard_input_group, textarea);
```

4.5.3 Driver Display/Input OpenGL ESr

Panoramica

Il driver display/Input **OpenGL ES** offre supporto per la simulazione del display LVGL e degli input da tastiera/mouse in una finestra del desktop creata tramite GLFW.
È un'alternativa a **Wayland**, **XCB**, **SDL** o **Qt**.

Lo scopo principale di questo driver è testare/eseguire il debug dell'applicazione LVGL in una finestra di simulazione **OpenGL**.

Prerequisiti

Il driver OpenGL utilizza GLEW GLFW per accedere al gestore finestre OpenGL.

1. Installare GLEW e GLFW: `sudo apt-get install libglew-dev libglfw3-dev`

Configurare il driver OpenGL

1. Librerie linkate richieste: -lGL -lGLEW -lglfw
2. **Abilitare il supporto del driver OpenGL in lv_conf.h, tramite la definizione del compilatore cmake o tramite KConfig**

```
#define LV_USE_OPENGLES  1
```

Utilizzo di Base

```
#include "lvgl/lvgl.h"
#include "lvgl/examples/lv_examples.h"
#include "lvgl/demos/lv_demos.h"

#define WIDTH 640
#define HEIGHT 480

int main()
{
    /* initialize lvgl */
    lv_init();

    /* create a window and initialize OpenGL */
    lv_glfw_window_t * window = lv_glfw_window_create(WIDTH, HEIGHT, true);

    /* create a display that flushes to a texture */
    lv_display_t * texture = lv_opengles_texture_create(WIDTH, HEIGHT);
    lv_display_set_default(texture);

    /* add the texture to the window */
    unsigned int texture_id = lv_opengles_texture_get_texture_id(texture);
    lv_glfw_texture_t * window_texture = lv_glfw_window_add_texture(window,
↪ texture_id, WIDTH, HEIGHT);

    /* get the mouse indev of the window texture */
    lv_indev_t * mouse = lv_glfw_texture_get_mouse_indev(window_texture);

    /* add a cursor to the mouse indev */
    LV_IMAGE_DECLARE(mouse_cursor_icon);
    lv_obj_t * cursor_obj = lv_image_create(lv_screen_active());
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_image_set_src(cursor_obj, &mouse_cursor_icon);
    lv_indev_set_cursor(mouse, cursor_obj);

    /* create Widgets on the screen */
    lv_demo_widgets();

    while (1)
    {
        uint32_t time_until_next = lv_timer_handler();
        if(time_until_next == LV_NO_TIMER_READY) time_until_next = LV_DEF_
↪REFR_PERIOD; /*handle LV_NO_TIMER_READY. Another option is to `sleep`␣
↪for longer*/
        lv_delay_ms(time_until_next);
    }

    return 0;
}
```

Utilizzo Avanzato

Il driver OpenGL può disegnare le texture dall'utente. È possibile utilizzare una libreria di terze parti per aggiungere contenuto a una texture e il driver disegnerà la texture nella finestra.

```
#include "lvgl/lvgl.h"
#include <GL/glew.h>
#include <GLFW/glfw3.h>

#define WIDTH 640
#define HEIGHT 480

void custom_texture_example(void)
{
    /*****************
    *  MAIN WINDOW
    *****************/

    /* create a window and initialize OpenGL */
    /* multiple windows can be created */
    lv_glfw_window_t * window = lv_glfw_window_create(WIDTH, HEIGHT, true);

    /****************************
    *   OPTIONAL MAIN TEXTURE
    ****************************/

    /* create a main display that flushes to a texture */
    lv_display_t * main_texture = lv_opengles_texture_create(WIDTH,␣
↪HEIGHT);
    lv_display_set_default(main_texture);

    /* add the main texture to the window */
    unsigned int main_texture_id = lv_opengles_texture_get_texture_id(main_
↪texture);
    lv_glfw_texture_t * window_main_texture = lv_glfw_window_add_
↪texture(window, main_texture_id, WIDTH, HEIGHT);

    /* get the mouse indev of this main texture */
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_indev_t * main_texture_mouse = lv_glfw_texture_get_mouse_
↪indev(window_main_texture);

    /* add a cursor to the mouse indev */
    LV_IMAGE_DECLARE(mouse_cursor_icon);
    lv_obj_t * cursor_obj = lv_image_create(lv_screen_active());
    lv_image_set_src(cursor_obj, &mouse_cursor_icon);
    lv_indev_set_cursor(main_texture_mouse, cursor_obj);

    /* create Widgets on the screen of the main texture */
    lv_demo_widgets();

    /**********************
    *   ANOTHER TEXTURE
    **********************/

    /* create a sub display that flushes to a texture */
    const int32_t sub_texture_w = 300;
    const int32_t sub_texture_h = 300;
    lv_display_t * sub_texture = lv_opengles_texture_create(sub_texture_w,␣
↪sub_texture_h);

    /* add the sub texture to the window */
    unsigned int sub_texture_id = lv_opengles_texture_get_texture_id(sub_
↪texture);
    lv_glfw_texture_t * window_sub_texture = lv_glfw_window_add_
↪texture(window, sub_texture_id, sub_texture_w, sub_texture_h);

    /* create Widgets on the screen of the sub texture */
    lv_display_set_default(sub_texture);
    lv_example_keyboard_2();
    lv_display_set_default(main_texture);

    /* position the sub texture within the window */
    lv_glfw_texture_set_x(window_sub_texture, 250);
    lv_glfw_texture_set_y(window_sub_texture, 150);

    /* optionally change the opacity of the sub texture */
    lv_glfw_texture_set_opa(window_sub_texture, LV_OPA_80);

    /*********************************************
    *   USE AN EXTERNAL OPENGL TEXTURE IN LVGL
    *********************************************/

    unsigned int external_texture_id;
    glGenTextures(1, &external_texture_id);
    glBindTexture(GL_TEXTURE_2D, external_texture_id);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    LV_IMAGE_DECLARE(img_cogwheel_argb);
#if LV_COLOR_DEPTH == 8
    const int texture_format = GL_R8;
#elif LV_COLOR_DEPTH == 16
```

(continues on next page)

(continua dalla pagina precedente)

```
    const int texture_format = GL_RGB565;
#elif LV_COLOR_DEPTH == 24
    const int texture_format = GL_RGB;
#elif LV_COLOR_DEPTH == 32
    const int texture_format = GL_RGBA;
#else
#error("Unsupported color format")
#endif
    glTexImage2D(GL_TEXTURE_2D, 0, texture_format, img_cogwheel_argb.
↪header.w, img_cogwheel_argb.header.h, 0, GL_BGRA, GL_UNSIGNED_BYTE, img_
↪cogwheel_argb.data);
    glGenerateMipmap(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, 0);

    /* add the external texture to the window */
    lv_glfw_texture_t * window_external_texture = lv_glfw_window_add_
↪texture(window, external_texture_id, img_cogwheel_argb.header.w, img_
↪cogwheel_argb.header.h);

    /* set the position and opacity of the external texture within the␣
↪window */
    lv_glfw_texture_set_x(window_external_texture, 20);
    lv_glfw_texture_set_y(window_external_texture, 20);
    lv_glfw_texture_set_opa(window_external_texture, LV_OPA_70);

    /*********************************************
    *   USE AN LVGL TEXTURE IN ANOTHER LIBRARY
    *********************************************/

    lv_refr_now(sub_texture);

    /* the texture is drawn on by LVGL and can be used by anything that␣
↪uses OpenGL textures */
    third_party_lib_use_texture(sub_texture_id);
}
```

Renderer con Cache delle Texture OpenGL

In LVGL è presente un renderer che memorizza nella cache le aree renderizzate tramite software come texture OpenGL. Le texture vengono recuperate dalla cache e riutilizzate quando viene rilevata una corrispondenza. Nella maggior parte dei casi, le prestazioni saranno notevolmente migliorate.

```
#define LV_USE_DRAW_OPENGLES 1
```

Limitazioni Note

- Le prestazioni saranno le stesse o leggermente peggiori se le aree disegnate non vengono mai trovate nella cache a causa di widget con colori o forme che variano continuamente. Un esempio è un'etichetta il cui colore è impostato su un valore casuale per ogni fotogramma, come nella scena "Multiple labels" della demo del benchmark.
- I layer con pixel trasparenti e una trasparenza complessiva del layer non si fonderanno correttamente. L'effetto può essere osservato nella scena "Containers with opa_layer" della demo del benchmark negli angoli del bordo.
- I layer con rotazione non sono attualmente supportati. Le immagini con rotazione funzionano correttamente.

4.5.4 Touchpad

Driver Linux Evdev

Panoramica

"Event device" (evdev) di Linux, è un'API indipendente dall'hardware che consente l'accesso agli eventi di input, ad esempio da un mouse o da un touchscreen. Viene esposto tramite l'interfaccia del file system del device Linux.

Prerequisiti

Il sistema dispone di un dispositivo di input configurato (solitamente in `/dev/input/`, ad esempio `/dev/input/event0`).

Configurazione del driver

Abilitare il supporto del driver evdev LVGL Linux in `lv_conf.h`.

```
#define LV_USE_EVDEV 1
```

Utilizzo

Per impostare un input di evento, creare prima un "input device" con `lv_edev_create` impostandolo sul dispositivo di eventi Linux corretto. Quindi collegarlo al display LVGL con `lv_indev_set_display`.

```
lv_indev_t *touch = lv_evdev_create(LV_INDEV_TYPE_POINTER, "/dev/input/
↪event0");
lv_indev_set_display(touch, disp);
```

Assicurarsi che un oggetto `lv_display_t` sia già stato creato per `disp`. Un esempio è mostrato di seguito, utilizzando il driver framebuffer Linux.

```
lv_display_t * disp = lv_linux_fbdev
lv_linux_fbdev_set_file(disp, "/dev/fb0");_create();
```

Individuazione del dispositivo di input

Se non si riesce a determinare il dispositivo di input, eseguire prima

`` `$cat /proc/bus/input/devices` ``

Questo dovrebbe mostrare i dispositivi di input e ci saranno voci con la parola `event` che forniscono un indizio sul dispositivo da utilizzare, ad esempio `event1` sarebbe `/dev/input/event1`.

Si può usare `evtest` per visualizzare i dati da quella sorgente di eventi e verificare se è effettivamente quella desiderata.

Provare:

`$evtest /dev/input/event1` sostituendo `eventX` con il dispositivo di eventi indicato sopra.

Rilevamento automatico dei dispositivi di input

È disponibile il supporto per la ricerca e l'aggiunta automatica di dispositivi di input in `/dev/input/`. I nuovi dispositivi verranno aggiunti automaticamente quando vengono connessi. Per abilitare questa funzionalità, è sufficiente chiamare `lv_evdev_discovery_start(NULL, NULL)`.

Si potrebbe voler reagire all'aggiunta di un nuovo dispositivo in modo che possa essere applicata un'immagine del cursore, ad esempio. È possibile fornire una funzione di callback che verrà richiamata quando viene aggiunto un nuovo dispositivo.

```
#include "lvgl/src/core/lv_global.h"

static void indev_deleted_cb(lv_event_t * e)
{
    if(LV_GLOBAL_DEFAULT()->deinit_in_progress) return;
    lv_obj_t * cursor_obj = lv_event_get_user_data(e)
    lv_obj_delete(cursor_obj);
}

static void discovery_cb(lv_indev_t * indev, lv_evdev_type_t type, void *␣
↪user_data)
{
    LV_LOG_USER("new '%s' device discovered", type == LV_EVDEV_TYPE_REL ?
↪"REL" :
                                              type == LV_EVDEV_TYPE_ABS ?
↪"ABS" :
                                              type == LV_EVDEV_TYPE_KEY ?
↪"KEY" :
                                              "unknown");

    if(type == LV_EVDEV_TYPE_REL) {
        LV_IMAGE_DECLARE(mouse_cursor_icon);
        lv_obj_t * cursor_obj = lv_image_create(lv_screen_active());
        lv_image_set_src(cursor_obj, &mouse_cursor_icon);
        lv_indev_set_cursor(indev, cursor_obj);
        lv_indev_add_event_cb(indev, indev_deleted_cb, LV_EVENT_DELETE,␣
↪cursor_obj);
    }
}

int main()
{
    /* ... */
    lv_evdev_discovery_start(discovery_cb, NULL);
    /* ... */
}
```

Al momento della stesura di questo documento, questa funzionalità non è supportata in BSD.

FT6X36

TODO

4.5.5 Driver per Display/Input Wayland

Panoramica

Il driver **Wayland** offre supporto per la simulazione del display LVGL e degli input da tastiera/mouse in una finestra del desktop.
È un'alternativa a **X11** o a **SDL2**

Lo scopo principale di questo driver è testare/eseguire il debug dell'applicazione LVGL; può anche essere utilizzato per eseguire applicazioni in 'kiosk mode'.

Dipendenze

Il driver Wayland richiede alcune dipendenze.

Su Ubuntu

```
sudo apt-get install libwayland-dev libxkbcommon-dev libwayland-bin␣
↪wayland-protocols
```

Su Fedora

```
sudo dnf install wayland-devel libxkbcommon-devel wayland-utils wayland-
↪protocols-devel
```

Configurazione del driver Wayland

1. Abilitare il driver Wayland in `lv_conf.h`

```
#define LV_USE_WAYLAND  1
```

2. Opzioni di configurazione facoltative:

Alcune impostazioni opzionali dipendono dall'abilitazione del supporto del buffer DMA (`LV_WAYLAND_USE_DMABUF`). La tabella seguente riassume le combinazioni valide e le relative limitazioni:

Tabella 2: Possibilità di configurazione

Opzione di configurazione	Senza DMABUF	Con DMABUF
`LV_DRAW_USE_G2D`	Non richiesto	**Obbligatorio**
`LV_WAYLAND_BUF_COUNT`	`1`	`1` o `2`
`LV_WAYLAND_RENDER_MODE`	`LV_DISPLAY_RENDER_MODE_PARTIAL`	`LV_DISPLAY_RENDER_MODE_DIRECT` o `LV_DISPLAY_RENDER_MODE_FULL`
`LV_WAYLAND_WINDOW_DECORATIONS`	`1` o `0`	`0`
`LV_WAYLAND_WL_SHELL` (**Obsoleto**)	Facoltativo (`1` o `0`)	Facoltativo (`1` o `0`)

Note aggiuntive

- Il supporto DMABUF (`LV_WAYLAND_USE_DMABUF`) migliora le prestazioni e abilita più modalità di rendering, ma presenta requisiti e restrizioni specifici.
- `LV_WAYLAND_WINDOW_DECORATIONS` è richiesto solo per alcuni compositori (ad esempio, GNOME/Mutter o Weston).
- `LV_WAYLAND_WL_SHELL` Abilita il supporto per `wl_shell` ormai obsoleto. Utile solo quando il BSP sul target ha weston `9.x`. (**Obsoleto** e verrà rimosso in una versione futura)

Esempio

Un simulatore di esempio è disponibile in questo repo

Utilizzo

1. In `main.c` `#include "lv_drivers/wayland/wayland.h"`
2. Abilitare il driver Wayland in `lv_conf.h` con `LV_USE_WAYLAND 1`
3. `LV_COLOR_DEPTH` dovrebbe essere impostato su `32` o su `16` in `lv_conf.h`
4. Aggiungere un display utilizzando `lv_wayland_window_create()`, possibilmente con una callback di chiusura per tracciare lo stato di ciascun display:

```
#define H_RES (800)
#define V_RES (480)

/* Create a display */
lv_disp_t * disp = lv_wayland_create_window(H_RES, V_RES, "Window Title",␣
↪close_cb);
```

Come parte della chiamata precedente, il driver Wayland registrerà quattro dispositivi di input per ciascun display:

- un KEYPAD connesso agli eventi della tastiera Wayland
- un POINTER connesso agli eventi touch Wayland
- un POINTER connesso agli eventi del puntatore Wayland
- un ENCODER connesso agli eventi degli assi del puntatore Wayland

Gli handle per i dispositivi di input di ciascun display possono essere ottenuti utilizzando rispettivamente `lv_wayland_get_indev_keyboard()`, `lv_wayland_get_indev_touchscreen()`, `lv_wayland_get_indev_pointer()` e `lv_wayland_get_indev_pointeraxis()`.

Modalità fullscreen

Per impostare la finestra a schermo intero tramite programmazione, utilizzare la funzione `lv_wayland_window_set_fullscreen()` rispettivamente con `true` o `false` per l'argomento `fullscreen`.

Modalità massimizzata

Per massimizzare la finestra tramite programmazione, utilizzare la funzione `lv_wayland_window_set_maximized()` rispettivamente con `true` o `false` per l'argomento `maximized`.

Gestore timer personalizzato

Chiamare sempre `lv_wayland_timer_handler()` nel proprio loop del timer invece del normale `lv_timer_handler()`.

Nota: `lv_wayland_timer_handler()` chiama internamente `lv_timer_handler()`

Questo consente al client Wayland di funzionare correttamente su Weston; il ridimensionamento dei buffer di memoria condivisa durante un commit non funziona correttamente su Weston.

Il wrapping della chiamata a `lv_timer_handler()` è necessario per avere un maggiore controllo su quando viene chiamata la callback di flush LVGL.

Building del driver Wayland

Un simulatore di esempio è disponibile in questo repo

Se è necessario utilizzare il driver con un altro sistema di build. I file sorgente e fli header per la shell XDG devono essere generati dalle definizioni per il protocollo della shell XDG.

Nell'esempio, CMake viene utilizzato per eseguire l'operazione richiamando l'utility `wayland-scanner`

Per eseguire questa operazione manualmente,

Assicurarsi che le dipendenze elencate all'inizio dell'articolo siano installate.

Il protocollo Wayland è definito utilizzando file XML presenti in `/usr/share/wayland-protocols`

Per generare i file richiesti, eseguire i seguenti comandi:

```
wayland-scanner client-header $SYSROOT/usr/share/wayland-protocols/stable/
↪xdg-shell/xdg-shell.xml wayland_xdg_shell.h
wayland-scanner private-code $SYSROOT/usr/share/wayland-protocols/stable/
↪xdg-shell/xdg-shell.xml wayland_xdg_shell.c
```

Quando `LV_WAYLAND_USE_DMABUF` è impostato a `1`, devono essere generati anche i seguenti protocolli:

```
wayland-scanner client-header $SYSROOT/usr/share/wayland-protocols/stable/
↪linux-dmabuf/linux-dmabuf-v1.xml wayland_linux_dmabuf.h
wayland-scanner private-code $SYSROOT/usr/share/wayland-protocols/stable/
↪linux-dmabuf/linux-dmabuf-v1.xml wayland_linux_dmabuf.c
```

I file risultanti possono quindi essere integrati nel progetto. È consigliabile eseguire nuovamente `wayland-scanner` su ogni build per garantire che vengano generate le versioni corrette, che devono corrispondere alla versione della libreria a "dynamically linked" `wayland-client` installata sul sistema.

Stato attuale e obiettivi

- Aggiungere la modalità di rendering diretto
- Refactoring delle integrazioni della shell per evitare un'eccessiva compilazione condizionale
- Tecnicamente, il driver Wayland consente di creare più finestre, ma questa funzionalità è sperimentale.
- Aggiungere in futuro un supporto avanzato per la shell XDG per consentire la creazione di applicazioni desktop su piattaforme Unix-like, in modo simile a quanto avviene con il driver Win32.
- Aggiungere il supporto per Mesa; attualmente viene utilizzato wl_shm, ma non è la tecnica più efficace.

Segnalazioni di bug

Il driver Wayland è attualmente in fase di sviluppo, segnalazioni di bug, contributi e feedback sono sempre benvenuti.

È tuttavia importante creare problemi dettagliati quando si riscontra un problema; log e screenshot del problema sono di grande aiuto.

Abilitare `LV_USE_LOG` e avviare l'eseguibile del simulatore in questo modo.

```
WAYLAND_DEBUG=1 ./path/to/simulator_executable > /tmp/debug 2>&1
```

Questo creerà un file di log chiamato `debug` nella directory `/tmp`, eseguire il copia e incolla del contenuto del file nell'issue su GitHub. Il file di log contiene i log LVGL e i messaggi Wayland.

Assicurarsi di replicare il problema rapidamente, altrimenti i log diventano troppo grandi.

4.5.6 Driver Display/Input di Windows

Panoramica

Il driver display/Input di **Windows** offre supporto per la simulazione del display LVGL e degli input da tastiera/mouse in una finestra Win32 di Windows.

Lo scopo principale di questo driver è testare/debuggare l'applicazione LVGL in una finestra di simulazione di **Windows** tramite la **simulator mode**, o sviluppare un'applicazione desktop standard di **Windows** con LVGL tramite la **application mode**.

Queste sono le **somiglianze** tra la modalità simulatore e la modalità applicazione.

- Supporto per l'integrazione di dispositivi LVGL per puntatore, tastiera e encoder.
- Supporto per l'input touch di Windows.
- Supporto per l'integrazione del metodo di input di Windows.
- Supporto per il riconoscimento DPI per monitor (sia V1 che V2).
- Fornire interoperabilità basata su HWND per altre infrastrutture dell'interfaccia utente di Windows.

Queste sono le **differenze** tra la modalità simulatore e la modalità applicazione.

Modalità Simulatore

- Progettata per lo scenario di simulazione di dispositivi LVGL: simula il rendering LVGL su un pannello di display hardware.
- Mantiene costante la risoluzione del display LVGL per simulare al meglio il layout dell'interfaccia utente che verrà visualizzato sui dispositivi di produzione.
- Quando si modifica l'impostazione di ridimensionamento DPI di Windows, il backend di Windows estenderà il contenuto del display.

Modalità Applicazione

- Progettata per lo scenario di sviluppo di applicazioni desktop Windows.
- Supporta il ridimensionamento delle finestre e la risoluzione del display LVGL viene modificata dinamicamente.
- Quando si modifica l'impostazione di ridimensionamento DPI di Windows, viene modificato anche il valore DPI del display LVGL.
- La risoluzione impostata per lv_windows_create_display corrisponde alla dimensione della finestra anziché alla dimensione del client della finestra, per seguire le convenzioni di altre infrastrutture di interfaccia utente desktop di Windows.
- Le applicazioni basate su questa modalità dovrebbero adattare la modifica della risoluzione del display LVGL per supportare correttamente il ridimensionamento delle finestre.

Prerequisiti

Il requisito minimo del sistema operativo Windows per questo driver è Windows Vista RTM.

Se si utilizzano librerie shim API Windows come YY-Thunks, il requisito minimo del sistema operativo Windows testato per questo driver è Windows XP RTM.

In base all'API GDI di Windows utilizzata da questo driver, è possibile che il requisito minimo del sistema operativo Windows per questo driver sia Windows 2000 RTM.

Configurare il Driver Windows

Abilitare il supporto del driver Windows in lv_conf.h, tramite la definizione del compilatore cmake o tramite KConfig

```
#define LV_USE_WINDOWS  1
```

Utilizzo

```
#include <Windows.h>
#include "lvgl/lvgl.h"
#include "lvgl/examples/lv_examples.h"
#include "lvgl/demos/lv_demos.h"

int main()
{
    lv_init();

    int32_t zoom_level = 100;
    bool allow_dpi_override = false;
    bool simulator_mode = false;
    lv_display_t* display = lv_windows_create_display(
        L"LVGL Display Window",
        800,
        480,
```

(continues on next page)

(continua dalla pagina precedente)

```
        zoom_level,
        allow_dpi_override,
        simulator_mode);
    if (!display)
    {
        return -1;
    }

    lv_lock();

    lv_indev_t* pointer_device = lv_windows_acquire_pointer_indev(display);
    if (!pointer_device)
    {
        return -1;
    }

    lv_indev_t* keypad_device = lv_windows_acquire_keypad_indev(display);
    if (!keypad_device)
    {
        return -1;
    }

    lv_indev_t* encoder_device = lv_windows_acquire_encoder_indev(display);
    if (!encoder_device)
    {
        return -1;
    }

    lv_demo_widgets();

    lv_unlock();

    while (1)
    {
        uint32_t time_till_next = lv_timer_handler();
        if(time_till_next == LV_NO_TIMER_READY) time_till_next = LV_DEF_
↪REFR_PERIOD; /*handle LV_NO_TIMER_READY. Another option is to `sleep`␣
↪for longer*/
        lv_delay_ms(time_till_next);
    }

    return 0;
}
```

4.5.7 Driver Display/Input X11

Panoramica

Il driver Display/Input **X11** offre supporto per la simulazione del display LVGL e degli input da tastiera/mouse in una finestra del desktop X11.
È un'alternativa a **Wayland**, **XCB**, **SDL** o **Qt**.

Lo scopo principale di questo driver è testare/eseguire il debug dell'applicazione LVGL in una finestra di simulazione **Linux**.

Prerequisiti

Il driver X11 utilizza XLib per accedere al gestore delle finestre Linux.

1. Installare XLib: `sudo apt-get install libx11-6` (dovrebbe essere già installato)
2. Installare il pacchetto di sviluppo XLib: `sudo apt-get install libx11-dev`

Configurare il driver X11

1. **Abilitare il supporto del driver X11 in lv_conf.h, tramite la definizione del compilatore cmake o tramite KConfig**

```
#define LV_USE_X11  1
```

2. **Opzioni di configurazione facoltative:**

   - **Direct Exit**

```
#define LV_X11_DIRECT_EXIT  1 /* preferred default - ends␣
↪the application automatically if last window has been␣
↪closed */
// or
#define LV_X11_DIRECT_EXIT  0 /* application is responsible␣
↪for ending the application (e.g. by own LV_EVENT_DELETE␣
↪handler */
```

   - **Double buffering**

```
#define LV_X11_DOUBLE_BUFFER  1 /* preferred default */
// or
#define LV_X11_DOUBLE_BUFFER  0 /* not recommended */
```

   - **Render mode**

```
#define LV_X11_RENDER_MODE_PARTIAL 1  /* LV_DISPLAY_RENDER_
↪MODE_PARTIAL, preferred default */
// or
#define LV_X11_RENDER_MODE_DIRECT  1  /* LV_DISPLAY_RENDER_
↪MODE_DIRECT, not recommended for X11 driver */
// or
#define LV_X11_RENDER_MODE_DULL    1  /* LV_DISPLAY_RENDER_
↪MODE_FULL, not recommended for X11 driver */
```

Utilizzo

Inizializzazione minima che apre una finestra e abilita il supporto per tastiera/mouse
(ad esempio, in main.c, LV_X11_DIRECT_EXIT dev'essere 1):

```
int main(int argc, char ** argv)
{
    ...

    /* initialize X11 display driver */
    lv_display_t * disp = lv_x11_window_create("LVGL X11 Simulation",␣
↪monitor_hor_res, monitor_ver_res);

    /* initialize X11 input drivers (for keyboard, mouse & mousewheel) */
    lv_x11_inputs_create(disp, NULL);
```

(continues on next page)

(continua dalla pagina precedente)

```
    ...

    while(true)
    {
        ...

        /* Periodically call the lv_timer handler */
        lv_timer_handler();
    }
}
```

Inizializzazione completa con simbolo del puntatore del mouse e gestione dell'uscita dall'applicazione (dipendente da LV_X11_DIRECT_EXIT (può essere 1 o 0))

```
bool terminated = false;

#if !LV_X11_DIRECT_EXIT
static void on_close_cb(lv_event_t * e)
{
    ...

    terminate = true;
}
#endif

int main(int argc, char ** argv)
{
    ...

    /* initialize X11 display driver */
    lv_display_t * disp = lv_x11_window_create("LVGL X11 Simulation",␣
↪monitor_hor_res, monitor_ver_res);
    lv_display_add_event_cb(disp, on_close_cb, LV_EVENT_DELETE, disp);

    /* initialize X11 input drivers (for keyboard, mouse & mousewheel) */
    LV_IMAGE_DECLARE(my_mouse_cursor_icon);
    lv_x11_inputs_create(disp, &my_mouse_cursor_icon);

    #if !LV_X11_DIRECT_EXIT
    /* set optional window close callback to enable application cleanup␣
↪and exit */
    lv_x11_window_set_close_cb(disp, on_close_cb, disp);
    #endif

    ...

    while(!terminated)
    {
        ...

        /* Periodically call the lv_timer handler */
        lv_timer_handler();
```

(continues on next page)

(continua dalla pagina precedente)

```
    }
}
```

4.5.8 Driver per Display/Input UEFI

Panoramica

Il driver display/input di **UEFI** offre supporto per l'utilizzo di LVGL con UEFI.

Prerequisiti

Sono necessari i seguenti protocolli disponibili:

- *EFI_LOADED_IMAGE_PROTOCOL_GUID*, per il supporto del file system (utilizzato per determinare il file system utilizzato per caricare l'applicazione)
- *EFI_SIMPLE_FILE_SYSTEM_PROTOCOL_GUID*, per il supporto del file system
- *EFI_SIMPLE_TEXT_INPUT_EX_PROTOCOL_GUID*, per il supporto della tastiera
- *EFI_SIMPLE_POINTER_PROTOCOL_GUID*, per il supporto del mouse
- *EFI_ABSOLUTE_POINTER_PROTOCOL_GUID*, per il supporto touch
- *EFI_GRAPHICS_OUTPUT_PROTOCOL_GUID*, per il disegno
- *EFI_EDID_ACTIVE_PROTOCOL_GUID*, per il disegno (facoltativo)

Configurare il driver UEFI

- Abilitare il supporto del driver UEFI in lv_conf.h

```
#define LV_USE_UEFI  1
```

- Abilitare le funzioni core della memoria, che sono wrapper per AllocatePool e FreePool (utilizzando memoria di tipo *EfiBootServicesData*) se non si desidera utilizzare le proprie implementazioni.

```
#define LV_UEFI_USE_MEMORY_SERVICES  1
```

- È possibile abilitare il supporto del file system per il file system da cui è stata caricata l'applicazione (lettera di default 'E')

```
#define LV_USE_FS_UEFI  1
```

- È necessario definire un file "include" che contenga le definizioni UEFI di base (protocolli e tipi); sono disponibili 2 file predefiniti che possono essere utilizzati per EDK2 e gnu-efi.

```
#define LV_USE_UEFI_INCLUDE <lvgl/src/drivers/uefi/lv_uefi_edk2.h>
// or
#define LV_USE_UEFI_INCLUDE <lvgl/src/drivers/uefi/lv_uefi_gnu_efi.h>
```

Utilizzo

```
#include "lvgl/lvgl.h"
#include "lvgl/examples/lv_examples.h"
#include "lvgl/demos/lv_demos.h"

EFI_STATUS EFIAPI EfiMain(EFI_HANDLE ImageHandle, EFI_SYSTEM_TABLE *␣
↪SystemTable)
{
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_uefi_init(ImageHandle, SystemTable);
    lv_init();

    if(!lv_is_initialized()) return EFI_NOT_READY;

    EFI_HANDLE handle = NULL;
    lv_display_t* display = NULL;
    lv_indev_t* indev = NULL;
    lv_group_t* group = NULL;
    lv_obj_t* cursor = NULL;
    // used to get out of the main loop
    size_t counter;

    // Init the display
    handle = lv_uefi_display_get_active();
    if(handle == NULL) {
        handle = lv_uefi_display_get_any();
    }
    if(handle == NULL) {
        lv_deinit();
        return EFI_UNSUPPORTED;
    }

    display = lv_uefi_display_create(handle);
    lv_display_set_default(display);

    // Create the group
    group = lv_group_create();
    lv_group_set_default(group);

    // Create an image that can be used as cursor
    cursor = lv_image_create(lv_layer_top());
    lv_image_set_src(cursor, "E:cursor.png");

    // Create the input devices
    indev = lv_uefi_simple_text_input_indev_create();
    lv_indev_set_group(indev, lv_group_get_default());
    lv_uefi_simple_text_input_indev_add_all(indev);

    indev = lv_uefi_simple_pointer_indev_create(NULL);
    lv_uefi_simple_pointer_indev_add_all(indev);
    lv_indev_set_cursor(indev, cursor);

    indev = lv_uefi_absolute_pointer_indev_create(NULL);
    lv_uefi_absolute_pointer_indev_add_all(indev);

    lv_demo_widgets();

    // Run main loop for ~ 10 seconds
    counter = 0;
    while(counter < 10000) {
        counter ++;
        gBS->Stall(1000);
        lv_tick_inc(1);
        lv_timer_handler();
    }
```

(continues on next page)

(continua dalla pagina precedente)

```
    return EFI_SUCCESS;
}
```

4.5.9 Driver SDL

Panoramica

SDL (Simple DirectMedia Layer) offre un modo multipiattaforma per gestire grafica, input e contenuti multimediali, rendendolo una scelta eccellente per l'esecuzione di applicazioni LVGL su PC.

Prerequisiti

Installare SDL in base alla piattaforma.

- Linux `sudo apt install libsdl2-dev`
- MacOS `brew install sdl2`
- Windows https://github.com/libsdl-org/SDL/releases

Configurare il driver SDL

1. Librerie linkate richieste: -lSDL2
2. Abilita il supporto del driver SDL in lv_conf.h, nelle definizioni del compilatore CMake o in KConfig.

```
#define LV_USE_SDL  1
#define LV_SDL_INCLUDE_PATH <SDL2/SDL.h>
#define SDL_HOR_RES 400
#define SDL_VER_RES 400
```

Utilizzo di Base

```
#include <unistd.h>
#define SDL_MAIN_HANDLED        /*To fix SDL's "undefined reference to␣
↪WinMain" issue*/
#include <SDL2/SDL.h>
#include "drivers/sdl/lv_sdl_mouse.h"
#include "drivers/sdl/lv_sdl_mousewheel.h"
#include "drivers/sdl/lv_sdl_keyboard.h"

static lv_display_t *lvDisplay;
static lv_indev_t *lvMouse;
static lv_indev_t *lvMouseWheel;
static lv_indev_t *lvKeyboard;

#if LV_USE_LOG != 0
static void lv_log_print_g_cb(lv_log_level_t level, const char * buf)
{
    LV_UNUSED(level);
    LV_UNUSED(buf);
}
#endif

int main()
{
```

(continues on next page)

(continua dalla pagina precedente)

```
    /* initialize lvgl */
    lv_init();

    // Workaround for sdl2 `-m32` crash
    // https://bugs.launchpad.net/ubuntu/+source/libsdl2/+bug/1775067/
↪comments/7
    #ifndef WIN32
        setenv("DBUS_FATAL_WARNINGS", "0", 1);
    #endif

    /* Register the log print callback */
    #if LV_USE_LOG != 0
    lv_log_register_print_cb(lv_log_print_g_cb);
    #endif

    /* Add a display
    * Use the 'monitor' driver which creates window on PC's monitor to␣
↪simulate a display*/

    lvDisplay = lv_sdl_window_create(SDL_HOR_RES, SDL_VER_RES);
    lvMouse = lv_sdl_mouse_create();
    lvMouseWheel = lv_sdl_mousewheel_create();
    lvKeyboard = lv_sdl_keyboard_create();

    /* create Widgets on the screen */
    lv_demo_widgets();


    Uint32 lastTick = SDL_GetTicks();
    while(1) {
        SDL_Delay(5);
        Uint32 current = SDL_GetTicks();
        lv_tick_inc(current - lastTick); // Update the tick timer. Tick is␣
↪new for LVGL 9
        lastTick = current;
        lv_timer_handler(); // Update the UI-
    }

    return 0;
}
```

Utilizzo di un IDE

LVGL con SDL è stato portato su vari IDE.

- Eclipse con driver SDL: Consigliato su Linux e Mac, supporta anche CMake
- VSCode con driver SDL: Consigliato su Linux (SDL) e Mac (SDL)
- Linux Generico: Progetto basato su CMake in cui è possibile passare facilmente da fbdev a DRM e SDL.

4.6 Renderer e GPU

4.6.1 Renderer Software

API

lv_draw_sw.h

lv_draw_sw_blend.h

lv_draw_sw_blend_private.h

lv_draw_sw_blend_to_al88.h

lv_draw_sw_blend_to_argb8888.h

lv_draw_sw_blend_to_argb8888_premultiplied.h

lv_draw_sw_blend_to_i1.h

lv_draw_sw_blend_to_l8.h

lv_draw_sw_blend_to_rgb565.h

lv_draw_sw_blend_to_rgb565_swapped.h

lv_draw_sw_blend_to_rgb888.h

lv_draw_sw_grad.h

lv_draw_sw_mask.h

lv_draw_sw_mask_private.h

lv_draw_sw_private.h

lv_draw_sw_utils.h

4.6.2 GPU Arm-2D

Arm-2D non è una GPU, ma **un livello di astrazione per GPU 2D dedicato ai microcontrollori**. Supporta tutti i processori Cortex-M, dal Cortex-M0 al più recente Cortex-M85.

Arm-2D accelera LVGL9 con due modalità: **Modalità Sincrona** e **Modalità Asincrona**.

- Quando sono disponibili **Helium** e **ACI (Arm Custom Instruction)**, si consiglia di utilizzare la **Modalità Sincrona** per accelerare LVGL.
- Quando sono disponibili GPU 2D basate su Arm-2D, ad esempio **GPU 2D basate su DMAC-350**, si consiglia di utilizzare la **Modalità Asincrona** per accelerare LVGL.

Arm-2D è un progetto open source su GitHub. Per ulteriori informazioni, consultare: https://github.com/ARM-software/Arm-2D.

Come si Usa

In generale:

- è possibile impostare la macro *`LV_USE_DRAW_ARM2D_SYNC`* a `1` e `LV_DRAW_SW_ASM` a `LV_DRAW_SW_ASM_HELIUM` in `lv_conf.h` per abilitare l'accelerazione sincrona Arm-2D per LVGL.
- È possibile impostare la macro `LV_USE_DRAW_ARM2D_ASYNC` a `1` in `lv_conf.h` per abilitare l'accelerazione asincrona Arm-2D per LVGL.

Se si utilizza CMSIS-Pack per il deploy di LVGL. Non è necessario definire manualmente la macro *`LV_USE_DRAW_ARM2D_SYNC`*, ma lv_conf_cmsis.h controllerà l'ambiente e imposterà *`LV_USE_DRAW_ARM2D_SYNC`* di conseguenza.

Considerazioni Progettuali

Come accennato in precedenza, Arm-2D è un layer di astrazione per GPU 2D; pertanto, se non è disponibile un acceleratore o un set di istruzioni dedicato (come Helium o ACI) per Arm-2D, l'aumento delle prestazioni per LVGL è trascurabile (a volte addirittura peggiore) per i normali processori Cortex-M.

Consigliamo vivamente di abilitare l'accelerazione Arm-2D per LVGL quando:

- I processori target sono **Cortex-M55**, **Cortex-M52** e **Cortex-M85**

- I processori target supportano Helium.
- Il fornitore del dispositivo fornisce un driver compatibile con ARM-2D per i propri acceleratori 2D proprietari e/o ACI (Arm Customized Instruction).
- Il dispositivo target contiene DMAC-350

Esempi

- Un progetto MDK basato su Cortex-M55 (supporta Helium), disponibile per l'emulazione PC.

API

lv_draw_sw_arm2d.h

lv_blend_arm2d.h

4.6.3 Acceleratore di Elaborazione Pixel Espressif

API

lv_draw_ppa.h

4.6.4 EVE External GPU Renderer

EVE is a type of external GPU IC which accepts high-level drawing commands over SPI and outputs the rendered graphics to a display over parallel RGB.

The advantage of using an EVE chip is that the rendering responsibility is removed from the driving MCU so it can be a lower-spec part or dedicate more of its processing time to other tasks. The SPI interface is simpler to connect than parallel RGB especially if the EVE chip is integrated into a display assembly.

LVGL features a renderer for EVE. LVGL UIs can be rendered by EVE and are effectively indistinguishable from the software renderer is most cases.

See also the *FT81x* framebuffer driver. It drives the same EVE chips but is a simpler, more standalone implementation which uses software rendering and sends all the pixels over SPI so it is much slower.

Limitazioni

- Image format, size, and count limit.
- Font format, size, and count limit.
- The total number of tasks rendered per refresh has an upper limit.
- Layers are not supported.

Utilizzo

Board Parameters

Find your display parameters and populate a `lv_draw_eve_parameters_t` struct with them.

Here is an example `lv_draw_eve_target_parameters.h` for the Riverdi RVT50HQBNWC00-B which has a BT817Q --- the EVE chip. These parameters were taken from the collection of configs provided by the EVE library supporting this renderer. See here. Check there for your board parameters.

```
lv_draw_eve_parameters_t params = {
    .hor_res = 800,
    .ver_res = 480,
    .vsync0 = 0,
    .vsync1 = 4,
    .voffset = 8,
```

(continues on next page)

(continua dalla pagina precedente)

```
    .vcycle = 496,
    .hsync0 = 0,
    .hsync1 = 4,
    .hoffset = 8,
    .hcycle = 816,
    .pclk = 3,
    .pclkpol = 1,
    .swizzle = 0,
    .cspread = 0,
    .has_crystal = true,
    .has_gt911 = false,
    .backlight_freq = 4000,
    .backlight_pwm = 128,
};
```

EVE Chip IO Implementation

The user is required to implement the GPIO and SPI IO functionality. The LVGL EVE renderer will call this callback to perform SPI communication with the EVE chip.

```
static void op_cb(lv_display_t * disp, lv_draw_eve_operation_t operation,
                  void * data, uint32_t length)
{
    /* optional: get the `user_data` parameter you passed to `lv_draw_eve_
↪display_create` */
    void * your_user_data = lv_draw_eve_display_get_user_data(disp);

    switch(operation) {
        case LV_DRAW_EVE_OPERATION_POWERDOWN_SET:
            /* setting the pin low powers down the EVE chip */
            your_gpio_write(PD_N_PIN, 0);
            break;
        case LV_DRAW_EVE_OPERATION_POWERDOWN_CLEAR:
            /* setting the pin high powers on the EVE chip */
            your_gpio_write(PD_N_PIN, 1);
            break;
        case LV_DRAW_EVE_OPERATION_CS_ASSERT:
            /* setting the pin low asserts the EVE chip SPI device */
            your_gpio_write(CS_N_PIN, 0);
            break;
        case LV_DRAW_EVE_OPERATION_CS_DEASSERT:
            /* setting the pin high de-asserts the EVE chip SPI device */
            your_gpio_write(CS_N_PIN, 1);
            break;
        case LV_DRAW_EVE_OPERATION_SPI_SEND:
            /* `data` is the data to send */
            your_spi_transmit(data, length);
            break;
        case LV_DRAW_EVE_OPERATION_SPI_RECEIVE:
            /* `data` is the destination for the data */
            your_spi_receive(data, length);
            break;
    }
}
```

You will also need to initialize your SPI peripheral and GPIO pins.

22 MHz was the highest SPI speed that worked during testing with the Riverdi board and the ESP32-S3. You may not have success with this speed so it is recommended to validate with an `SPI_SPEED` value of `10` (10 MHz) and increase experimentally in your testing.

LVGL EVE Display Creation

To create the LVGL display for the EVE renderer, you call *`lv_draw_eve_display_create`*`(params, op_cb, your_user_data)` which returns the created display. `your_user_data` can be `NULL`. It should be called after GPIO and SPI is initialized. You may choose to initialize your IO the first time `op_cb` is called.

No buffers are required for the LVGL EVE renderer because no pixels are written to any buffers in the device running LVGL. When something needs to be drawn, a series of commands are sent to EVE.

Touch Indev Creation

`lv_draw_eve_touch_create``(disp)` creates a touch *Input Device (lv_indev)* for the display.

You may need to configure the i2c address of the touch controller connected to EVE. See the section *EVE Register Access* for more info about register access.

Here is an example of setting the `REG_TOUCH_CONFIG` register on a BT817q EVE chip for a capacitive touch screen with a controller that has the i2c address `0x15`.

```
/*
15:   0: capacitive, 1: resistive         CAPACITIVE
14:   host mode                           NO
13:   reserved
12:   ignore short circuit protection     NO
11:   low-power mode                      NO
10-4: 7-bit i2c address                   0x15
 3:   reserved
 2:   suppress 300ms startup              NO
 1-0: 2-bit sampling clocks val           use 1 (the reset default)
*/
lv_draw_eve_memwrite16(disp, LV_EVE_REG_TOUCH_CONFIG, 0x0151);
```

Rotazione del Display

Efficient display rotation is fully supported through *`lv_display_set_rotation()`*. Touch input rotation is handled accordingly.

EVE Register Access

The functions *`lv_draw_eve_memread8()`*, *`lv_draw_eve_memread16()`*, *`lv_draw_eve_memread32()`*, *`lv_draw_eve_memwrite8()`*, *`lv_draw_eve_memwrite16()`*, *`lv_draw_eve_memwrite32()`* are available if needed. They are wrappers around `EVE_memRead8`, etc.

Register definitions and other EVE enumerations are available when you include `lvgl.h` under the prefix namespace `LV_EVE_`. I.e., `REG_ID` is available as `LV_EVE_REG_ID` and `EVE_ROM_CHIPID` is available as `LV_EVE_EVE_ROM_CHIPID`, etc.

Further Reading

- https://brtchip.com/wp-content/uploads/Support/Documentation/Programming_Guides/ICs/EVE/FT81X_Series_Programmer_Guide.pdf
- https://brtchip.com/wp-content/uploads/2024/06/BRT_AN_033_BT81X-Series-Programming-Guide.pdf

4.6.5 Accelerazione NemaGFX

NemaGFX è un'API grafica di alto livello supportata da molte GPU 2.5D embedded. LVGL può utilizzarla per il rendering nativo della grafica LVGL.

Iniziare a usare il display Riverdi STM32U5 da 5 pollici

lv_port_riverdi_stm32u5 è un "porting" pronto all'uso per il display embedded Riverdi STM32 da 5,0" (STM32U599NJH6Q o STM32U5A9NJH6Q) con supporto Nema. Seguire le istruzioni nel file readme per iniziare.

lv_port_stm32u5g9j-dk2 è un "porting" pronto all'uso per il devkit STM32U5G9J-DK2 con display da 5,0" e GPU NeoChrom VG. Seguire le istruzioni nel file readme per iniziare.

Utilizzo e Configurazione

Abilitare il renderer impostando *LV_USE_NEMA_GFX* a `1` in lv_conf.h. Se si utilizza `LV_USE_NEMA_VG`, impostare `LV_NEMA_GFX_MAX_RESX` e `LV_NEMA_GFX_MAX_RESY` in base alle dimensioni del display che si utilizzerà, in modo che venga riservata sufficiente memoria statica per VG. Senza VG, il renderer software eseguirà più tipi di attività.

"libs/nema_gfx" contiene i binari precompilati per i driver GPU Nema.

lv_port_riverdi_stm32u5 è già configurato per linkare i binari "cortex_m33_revC" durante la compilazione e lv_port_stm32u5g9j-dk2 configurato per linkare i binari "cortex_m33_NemaPVG" durante la compilazione.

"cortex_m33_revC" funziona su tutti i dispositivi STM32 m33 con core Nema, mentre "cortex_m33_NemaPVG" supporta inoltre l'accelerazione di tassellazione e moltiplicazione di matrici di cui sono capaci i core Nema U5F e U5G STM32.

Con un diverso progetto STM32CubeIDE, è possibile configurare le librerie da linkare cliccando col pulsante destro del mouse sul progetto nella barra laterale "Project Explorer", selezionando "Properties", andando su "C/C++ Build", "Settings", "MCU G++ Linker" e quindi su "Libraries". Aggiungere una voce in "Libraries (-l)" che sia "nemagfx-float-abi-hard". Aggiungere una voce in "Library search path (-L)" che sia un path per "libs/nema_gfx/lib/core/cortex_m33_revC/gcc" ad esempio "${workspace_loc:/${ProjName}/Middlewares/LVGL/lvgl/libs/nema_gfx/lib/core/cortex_m33_revC/gcc}". Aggiungere anche la directory "libs/nema_gfx/include" ai percorsi di ricerca degli include. In "MCU GCC Compiler", "Include paths", aggiungere una voce a "Include paths (-I)" che corrisponde al percorso di "libs/nema_gfx/include" ad esempio "${workspace_loc:/${ProjName}/Middlewares/LVGL/lvgl/libs/nema_gfx/include}". Cliccare su "Apply and Close".

> **Nota**
>
> `cannot find -lnemagfx-float-abi-hard: No such file or directory`
>
> La versione della libreria per ST è stata aggiornata a `revC` quindi il componente del path `cortex_m33` deve essere aggiornato a `cortex_m33_revC` in STM32CubeIDE.

È supportato *LV_COLOR_DEPTH* a 32 e 16 bit.

Al momento della stesura di questo articolo, il supporto per *LV_USE_OS* è sperimentale e non ancora funzionante in lv_port_riverdi_stm32u5

NemaGFX richiede una semplice implementazione HAL per allocare memoria ed eventualmente bloccare le risorse. È possibile utilizzare un'implementazione HAL personalizzata per la propria piattaforma oppure utilizzare una delle implementazioni fornite impostando `LV_USE_NEMA_HAL` su un valore diverso da *LV_NEMA_HAL_CUSTOM*.

Immagini TSC

Questo motore di rendering può disegnare immagini TSC (ThinkSilicon Compression). I formati colore TSC 4/6/6A/12/12A fanno parte di *lv_color_format_t*. Tutti gli altri motori di rendering ignoreranno le immagini con questi formati colore. Definire una variabile descrittore di immagine con il formato colore TSC corrispondente

e la GPU sarà in grado di disegnarla direttamente. Non è necessario specificare Stride perché verrà calcolato dal motore di rendering.

```
const lv_image_dsc_t img_demo_widgets_avatar_tsc6a = {
    .header.cf = LV_COLOR_FORMAT_NEMA_TSC6A,
    .header.w = 144,
    .header.h = 144,
    .data = img_demo_widgets_avatar_tsc6a_map,
    .data_size = sizeof(img_demo_widgets_avatar_tsc6a_map),
};
```

DMA2D

Il renderer Nema utilizza DMA2D per il flush in parallelo con il rendering in lv_port_riverdi_stm32u5.

Se l'STM non ha la GPU Nema, potrebbe comunque supportare DMA2D. DMA2D è una semplice periferica in grado di disegnare riempimenti e immagini indipendentemente dalla CPU. Vedere il *Supporto DMA2D* di LVGL.

API

lv_draw_nema_gfx.h

lv_draw_nema_gfx_utils.h .. Autogenerato

lv_nema_gfx_path.h

4.6.6 NXP PXP GPU

API

lv_draw_pxp.h

lv_pxp_cfg.h

4.6.7 NXP VGLite GPU

API

lv_draw_vglite.h

4.6.8 NXP G2D GPU

API

lv_draw_g2d.h

4.6.9 Renderer SDL

API

lv_draw_sdl.h

lv_sdl_keyboard.h

lv_sdl_mouse.h

lv_sdl_mousewheel.h

lv_sdl_private.h

lv_sdl_window.h

4.6.10 STM32 DMA2D GPU

API

lv_draw_dma2d.h

4.6.11 GPU Generica VG-Lite

Si tratta di un'implementazione generica del backend di rendering VG-Lite, progettata per utilizzare l'API generica di VeriSilicon per gestire l'hardware GPU il più possibile.

Anche con diversi produttori di chip, purché utilizzino la stessa versione dell'API VG-Lite come backend di rendering, l'accelerazione del rendering LVGL può essere supportata senza la necessità di adattamenti LVGL.

Configurazione

1. Impostare `LV_USE_DRAW_VG_LITE` a 1 in `lv_conf.h` per abilitare il backend di rendering VG-Lite. Assicurarsi che l'hardware sia stato adattato all'API VG-Lite e che sia stato esposto il path assoluto a `vg_lite.h`, a cui lvgl può fare riferimento direttamente.
2. Per confermare il metodo di inizializzazione della GPU, ci sono due modi:
   - L'SDK richiama autonomamente la funzione di inizializzazione della GPU durante l'avvio del sistema e la GPU è disponibile all'avvio di LVGL; impostare `LV_VG_LITE_USE_GPU_INIT` a 0.
   - LVGL richiama attivamente la funzione di inizializzazione della GPU e l'SDK deve implementare la funzione pubblica `gpu_init()`. LVGL la richiamerà per completare l'inizializzazione hardware della GPU durante l'avvio; impostare `LV_VG_LITE_USE_GPU_INIT` a 1.
3. Impostare la configurazione `LV_VG_LITE_USE_ASSERT` per abilitare il controllo dei parametri di chiamata della GPU. A causa della complessità dei parametri utilizzati nelle chiamate della GPU, parametri errati possono causare un funzionamento anomalo dell'hardware della GPU, ad esempio dimenticando di aggiungere un simbolo di fine al path o non rispettando i requisiti di allineamento per il buffer stride. Per risolvere rapidamente tali problemi, è stato aggiunto un rigoroso controllo dei parametri prima di ogni chiamata a VG-Lite, che include la convalida del buffer stride e il controllo dell'invertibilità della matrice. Quando viene rilevato un parametro di errore, verrà eseguita un'asserzione per visualizzarlo, consentendo all'utente di apportare tempestivamente correzioni e ridurre il tempo sprecato nella simulazione hardware. Si noti che l'abilitazione di questo controllo ridurrà le prestazioni del runtime. Si consiglia di abilitarlo in modalità Debug e di disabilitarlo nella versione Release.
4. Impostare la configurazione `LV_VG_LITE_FLUSH_MAX_COUNT` per specificare il metodo di "flush". VG-Lite utilizza due set di buffer di comando per il rendering delle istruzioni e un utilizzo efficace di questo meccanismo può migliorare notevolmente l'efficienza di disegno. Attualmente sono supportati due metodi di buffering:
   - Impostare `LV_VG_LITE_FLUSH_MAX_COUNT` a zero (consigliato). Il backend di rendering otterrà lo stato di funzionamento della GPU ogni volta che scrive istruzioni di rendering nel buffer di comando.

   Quando la GPU è inattiva, chiamerà immediatamente `vg_lite_flush` per notificare alla GPU di avviare il rendering e scambiare il buffer di comando. Quando la GPU è occupata, continuerà a riempire la cache del buffer di comando con le istruzioni di rendering. Il driver sottostante determinerà automaticamente se il buffer di comando è stato riempito. Quando sta per essere riempito, attenderà forzatamente la fine delle attività di disegno non completate e scambierà il buffer dei comandi. Questo metodo può migliorare efficacemente l'utilizzo della GPU, soprattutto in scenari in cui si esegue il rendering del testo, poiché il tempo di disegno della GPU e il tempo di preparazione dei dati della CPU sono molto simili, consentendo a CPU e GPU di funzionare in parallelo.
   - Impostare `LV_VG_LITE_FLUSH_MAX_COUNT` su un valore maggiore di zero, ad esempio 8. Dopo aver scritto 8 istruzioni di rendering nel buffer dei comandi, il backend di rendering

   chiamerà `vg_lite_flush` per notificare alla GPU di avviare il rendering e scambiare il buffer dei comandi.
5. Impostare la configurazione `LV_VG_LITE_USE_BOX_SHADOW` per utilizzare il rendering GPU per le ombreggiature. In effetti, l'hardware GPU non supporta effettivamente il rendering delle ombreggiature. Tuttavia, attraverso la sperimentazione, si è scoperto che un effetto ombra simile può essere ottenuto utilizzando più

layer di bordi con diversi livelli di trasparenza. Si consiglia di abilitare questa configurazione in scenari in cui i requisiti di qualità delle ombre non sono elevati, in quanto può migliorare significativamente l'efficienza di rendering.

6. Impostare la configurazione `LV_VG_LITE_GRAD_CACHE_CNT` per specificare il numero di voci della cache del gradiente. Il disegno di gradienti include gradienti lineari e radiali. L'utilizzo di una cache può ridurre efficacemente il numero di volte in cui l'immagine del gradiente viene creata e migliorare l'efficienza del disegno. Ogni singolo gradiente consuma circa 4 KB di memoria della GPU. Se nell'interfaccia vengono utilizzati molti gradienti, si può provare ad aumentare il numero di voci nella cache dei gradienti. Se l'API VG-Lite restituisce l'errore `VG_LITE_OUT_OF_RESOURCES`, si può provare ad aumentare la dimensione del pool di memoria della GPU o a ridurre il numero di voci nella cache dei gradienti.
7. Impostare la configurazione `LV_VG_LITE_STROKE_CACHE_CNT` per specificare il numero di cache del path del tratto. Se i parametri del tratto non cambiano, i parametri del tratto generati in precedenza vengono recuperati automaticamente dalla cache per migliorare le prestazioni di rendering. La memoria occupata dal tratto è strettamente correlata alla lunghezza del percorso. Se l'API VG-Lite restituisce l'errore `VG_LITE_OUT_OF_RESOURCES`, si può provare ad aumentare la dimensione del pool di memoria della GPU o a ridurre il numero di voci nella cache dei gradienti.

NOTA: Il backend di rendering VG-Lite non supporta chiamate multi-thread, assicurarsi che *`LV_USE_OS`* sia sempre configurato come *`LV_OS_NONE`*.

VG-Lite Simulator

LVGL integra un simulatore VG-Lite basato su ThorVG. Il suo scopo è semplificare il debug dell'adattamento VG-Lite e ridurre i tempi di debug e individuazione dei problemi sui dispositivi hardware. Per istruzioni dettagliate, vedere *VG-Lite Simulator*.

API

lv_draw_vglite.h

lv_vglite_buf.h

lv_vglite_matrix.h

lv_vglite_path.h

lv_vglite_utils.h

lv_vg_lite_decoder.h

lv_vg_lite_grad.h

lv_vg_lite_path.h

lv_vg_lite_pending.h

lv_vg_lite_stroke.h

lv_vg_lite_utils.h

4.7 Framework

4.7.1 Arduino

La libreria LVGL library è disponibile direttamente come libreria Arduino.

Si noti che è necessario scegliere una scheda sufficientemente potente per eseguire LVGL e la propria GUI. Vedere i *requisiti di LVGL*.

Ad esempio, ESP32 è un buon candidato per creare interfacce utente con LVGL.

Se si utilizza Arduino GIGA Display Shield (la documentazione qui), utilizzare la libreria Arduino_H7_Video, inclusa nel pacchetto della scheda Arduino Mbed OS (qui), che fornisce compatibilità LVGL integrata.

Ottenere la libreria LVGL per Arduino

LVGL può essere installato tramite l'Arduino IDE Library Manager o come libreria .ZIP.

Si può fare il Download dell'ultima versione di LVGL da GitHub e copiarla semplicemente nella cartella della libreria di Arduino.

Configurare i driver

Per iniziare, si consiglia di utilizzare la libreria TFT_eSPI come driver TFT per semplificare i test. Per farla funzionare, configurare `TFT_eSPI` in base al tipo di display TFT modificando:

- `User_Setup.h`
- oppure selezionando una configurazione in `User_Setup_Select.h`

Entrambi i file si trovano nella cartella della libreria `TFT_eSPI`.

Configurare LVGL

LVGL ha un proprio file di configurazione chiamato `lv_conf.h`. Una volta installato LVGL, seguire questi passaggi di configurazione:

1. Andare alla directory delle librerie installate di Arduino
2. Andare a `lvgl` e copiare `lv_conf_template.h` come `lv_conf.h` nella directory delle Librerie Arduino accanto alla cartella delle librerie di `lvgl`.
3. Aprire `lv_conf.h` e modificare il primo `#if 0` in `#if 1` per abilitare il contenuto del file
4. Impostare la profondità di colore del display in *`LV_COLOR_DEPTH`*

Infine, il layout con `lv_conf.h` dovrebbe apparire così:

```
arduino
 |-libraries
   |-lvgl
   |-other_lib_1
   |-other_lib_2
   |-lv_conf.h
```

Inizializzare ed eseguire LVGL

Consultare LVGL_Arduino.ino per vedere come inizializzare LVGL. `TFT_eSPI` viene utilizzato come driver del display.

Nel file INO si può vedere come registrare un display e un touchpad per LVGL e chiamare un esempio.

Usare gli esempi e le demo

Notare che non esiste un file INO dedicato per ogni esempio. Invece, si può caricare un esempio chiamando la funzione `lv_example_...`. Ad esempio `lv_example_btn_1()`.

importante
A causa di alcune limitazioni del sistema di compilazione di Arduino, è necessario copiare `lvgl/examples` in `lvgl/src/examples`. Analogamente, per le demo `lvgl/demos` in `lvgl/src/demos`.

Debug e logging

LVGL può visualizzare informazioni di debug in caso di problemi. Nell'esempio `LVGL_Arduino.ino` è presente il metodo `my_print`, che invia queste informazioni di debug all'interfaccia seriale. Per abilitare questa funzionalità, è necessario modificare il file `lv_conf.h` e abilitare il logging nella sezione `log settings`:

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

Dopo aver abilitato il modulo di log e impostato `LV_LOG_LEVEL` di conseguenza, il log di output viene inviato alla porta `Seriale` @ 115200 bps.

4.7.2 PlatformIO

TODO

4.7.3 Tasmota e berry

Cos'è Tasmota?

Tasmota è un firmware open source ampiamente utilizzato per dispositivi basati su ESP8266 ed EPS32. Supporta un'ampia varietà di dispositivi, sensori e integrazioni con servizi di domotica e cloud. Il firmware Tasmota viene scaricato più di 200.000 volte al mese e vanta una community attiva e in crescita.

Tasmota offre accesso a centinaia di dispositivi supportati, supporto completo di MQTT, HTTP(S), integrazione con i principali sistemi di domotica, una miriade di sensori, IR, RF, Zigbee, Bluetooth, AWS IoT, Azure IoT, Alexa e molto altro.

Cos'è Berry?

Berry è un linguaggio di scripting embedded a tipizzazione dinamica ultraleggero. È progettato per dispositivi embedded a basse prestazioni. L'interprete di Berry include un compilatore one-pass e una VM basata su registri; tutto il codice è scritto in ANSI C99. Berry offre una sintassi molto simile a Python e si ispira alla VM LUA. È completamente integrato in Tasmota.

Punti di forza di Berry

Berry offre i seguenti vantaggi:

- Leggero: Un interprete ben ottimizzato con risorse molto ridotte. Ideale per l'uso nei microprocessori.
- Veloce: compilatore bytecode ottimizzato a passaggio singolo e macchina virtuale basata su registri.
- Potente: supporta la programmazione imperativa, la programmazione orientata agli oggetti e la programmazione funzionale.
- Flessibile: Berry è uno script di tipo dinamico ed è progettato per l'integrazione nelle applicazioni. Può fornire una buona scalabilità dinamica per il sistema host.
- Semplice: sintassi semplice e naturale, supporto per la garbage collection e interfaccia FFI (foreign function interface) facile da usare.
- Risparmio di RAM: Con la costruzione di oggetti in fase di compilazione, la maggior parte degli oggetti costanti viene memorizzata in segmenti di dati di codice di sola lettura, quindi l'utilizzo di RAM dell'interprete è molto basso all'avvio.

Tutte le funzionalità sono descritte nel Berry Reference Manual

Perché LVGL + Tasmota + Berry?

Nel 2021, Tasmota ha aggiunto il supporto completo di LVGL per i dispositivi basati su ESP32. Ha inoltre introdotto il linguaggio di scripting Berry, un linguaggio compatto simile a Python e completamente integrato in Tasmota.

È ora disponibile una mappatura completa di LVGL nel linguaggio Berry, simile a quella di MicroPython. Consente di utilizzare oltre il 98% di tutte le funzionalità di LVGL. È anche possibile scrivere widget personalizzati in Berry.

Versioni supportate: LVGL v8.0.2, LodePNG v20201017, Freetype 2.10.4

Tasmota + Berry + LVGL possono essere utilizzati per:

- Interfaccia grafica per prototipazione rapida.
- Abbreviare il ciclo di modifica e messa a punto dell'interfaccia utente grafica.
- Modellare la GUI in modo più astratto definendo widget compositi riutilizzabili, sfruttando le funzionalità del linguaggio Berry come Ereditarietà, Chiusure, Gestione delle Eccezioni...
- Rendere LVGL accessibile a un pubblico più vasto. Non è necessario conoscere il C per creare una buona GUI su un sistema embedded.

È in fase di sviluppo anche un'interfaccia di livello superiore compatibile con OpenHASP.

Che aspetto ha?

In breve: Simile a MicroPython, è molto simile all'API C, ma orientata agli oggetti per i componenti LVGL.

Immergiamoci subito in un esempio!

Un semplice esempio

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

Come si usa?

Si può iniziare in meno di 10 minuti su un M5Stack o un dispositivo equivalente in meno di 10 minuti con questo breve tutorial

Dove si possono trovare maggiori informazioni?

- Documentazione di Tasmota
- Documentazione di Berry
- Documentazione di Berry per Tasmota LVGL

4.8 IDE

4.8.1 MDK

TODO

4.8.2 Simulatore su PC

Si può provare LVGL **utilizzando solo il PC** (ovvero senza alcuna scheda di sviluppo). LVGL verrà eseguito in un ambiente di simulazione sul PC, dove chiunque può scrivere e sperimentare applicazioni LVGL reali.

L'utilizzo del simulatore su un PC presenta i seguenti vantaggi:

- Indipendenza dall'hardware: si scrive il codice, lo si esegue sul PC e se ne visualizza il risultato su un monitor.
- Multipiattaforma: Qualsiasi sistema Windows, Linux o macOS può eseguire il simulatore per PC.
- Portabilità: Il codice scritto è portabile, il che significa che si può semplicemente copiarlo durante la migrazione su hardware embedded.
- Facile Validazione: Il simulatore è anche molto utile per segnalare bug perché fornisce una piattaforma comune per tutti gli utenti.
- Una migliore esperienza per gli sviluppatori: Su PC i debugger sono solitamente più veloci e migliori, si può loggare su file, aggiungere molte `printf`, eseguire la profilazione e così via.

Selezionare un IDE

Il simulatore è compatibile con diversi IDE (Integrated Development Environment). Scegli l'IDE preferito, leggerne il suo file README su GitHub, scaricare il progetto e caricarlo nell'IDE.

- Eclipse con driver SDL: Consigliato su Linux e Mac, supporta anche CMake
- VisualStudio: Consigliato su Windows
- VSCode con driver SDL: Consigliato su Linux (SDL) e Mac (SDL)
- CodeBlocks: Consigliato su Windows
- PlatformIO con driver SDL: Consigliato su Linux e Mac, ma supporta anche un ambiente STM32
- Linux Generico: Progetto basato su CMake in cui è possibile passare facilmente da fbdev a DRM e SDL.
- MDK con FastModel: Per Windows

Progetto esterno non gestito dall'organizzazione LVGL:

- QT Creator: Multipiattaforma

Driver integrati

LVGL include diversi *driver integrati*.

Anche se un progetto di simulazione include, ad esempio, SDL, è possibile sostituirlo facilmente abilitando un altro driver in `lv_conf.h` e chiamando la sua funzione `create`.

Ad esempio, per utilizzare il dispositivo frame buffer di Linux al posto di SDL, è sufficiente abilitare `LV_USE_LINUX_FBDEV` e chiamare

```
lv_display_t *display = lv_linux_fbdev_create();
lv_linux_fbdev_set_file(display, "/dev/fb0")
```

4.9 (RT)OS

4.9.1 Buildroot

Buildroot è un set di Makefile e patch che semplifica e automatizza il processo di creazione di un ambiente Linux completo e avviabile per un sistema embedded, utilizzando al contempo la cross-compilazione per consentire la creazione per più piattaforme target su un singolo sistema di sviluppo basato su Linux.

Configurazione Rapida

È disponibile un repository Git che include tutto il necessario per testare la configurazione di Buildroot senza seguire la guida. È pensato per scopi di test e può essere utilizzato se si ha già familiarità con i concetti e l'architettura di Buildroot.

Ottenere il repository

```
git clone --recurse-submodules https://github.com/lvgl/lv_buildroot.git
```

Creare l'immagine

```
./build.sh
```

A questo punto, si ha un'immagine che può essere flashata su una SD e utilizzata.

Ora compilrea l'SDK e installarlo

```
cd output
make sdk
mkdir -p ~/sdk
tar -xzf images/aarch64-buildroot-linux-gnu_sdk-buildroot.tar.gz -C ~/sdk
```

L'SDK è installato. Compilare l'applicazione.

```
cd ../application/lv_benchmark
source ../setup-build-env.sh
cmake -B build -S .
make -j $(nproc) -C build
cd ../..
```

Installare il binario in rootfs_overlay

```
cp application/lv_benchmark/bin/lvgl-app resources/board/rootfs_overlay/
↪usr/bin/
```

Rigenerare l'immagine con l'overlay rootfs aggiornato.

```
cd output
make
```

Andare su *Eseguire il flashing dell'immagine* per testare il sistema.

Immagine personalizzata RPi4

Questo capitolo offre una guida dettagliata per la creazione di un'immagine personalizzata per Raspberry Pi 4 (RPi4). Verranno introdotti i componenti e i concetti chiave di Buildroot. Il processo descritto qui può essere adattato per l'utilizzo con altre schede. Per funzionalità avanzate non trattate in questa guida, consultare il Manuale di Buildroot

Questa guida è stata realizzata su `Ubuntu 22.04`

Preparare l'area di lavoro

L'area di lavoro è strutturata per utilizzare Buildroot senza alcuna modifica. Eseguire i seguenti comandi:

```
mkdir rpi4_buildroot_lvgl && cd rpi4_buildroot_lvgl
mkdir resources && cd resources
mkdir board
mkdir configs
touch Config.in && touch external.desc && touch external.mk
echo "name: rpi4_lvgl" >> external.desc
```

(continues on next page)

(continua dalla pagina precedente)

```
echo "desc: rpi4 lvgl custom build" >> external.desc
cd ..
```

Ogni utilità di gestione cartelle verrà spiegata nel corso della guida.

Ottenere Buildroot

Innanzitutto, secondo il suo manuale, Buildroot richiede l'installazione di determinati pacchetti prima di avviare la compilazione. Installiamoli utilizzando il gestore pacchetti di Ubuntu.

```
sudo apt install sed make binutils gcc g++ bash patch gzip bzip2 perl tar \
        cpio python3 unzip rsync wget libncurses-dev
```

Ora che abbiamo soddisfatto tutti i prerequisiti, scarichiamo una versione stabile di Buildroot.

```
git clone --branch 2024.08 --depth 1 https://github.com/buildroot/
↪buildroot.git
```

Creare un'immagine RPi4

Vediamo se esiste già una configurazione per RPi4 in Buildroot:

```
cd buildroot
make list-defconfigs | grep rasp
```

È disponibile una build per RPi4 a 62 bit: `raspberrypi4_64_defconfig`.

Si possono trovare tutte le configurazioni anche nel repository Buildroot `buildroot > configs`

Iniziare con questa configurazione come base, ma deve essere modificata per soddisfare i requisiti di lvgl. Quindi copiamo questa base e teniamola fuori da Buildroot.

```
# move to the project root
cp buildroot/configs/raspberrypi4_64_defconfig resources/configs/
↪raspberrypi4_lvgl_defconfig
```

Creare un nuovo file `build.sh` nella radice del progetto con questo contenuto:

```
#!/bin/bash
SCRIPTPATH="$( cd "$(dirname "$0")" >/dev/null 2>&1 ; pwd -P )"

cd $SCRIPTPATH
mkdir -p output
cd buildroot
make defconfig O=../output BR2_DEFCONFIG=../resources/configs/raspberrypi4_
↪lvgl_defconfig
```

Rendere questo file eseguibile.

```
chmod +x build.sh
./build.sh
```

L'esecuzione di questo script crea una cartella di output che contiene tutto ciò che il processo genererà:

- Risorse recuperate
- Toolchain
- Immagini

Questo aiuta a mantenere pulita la directory Buildroot. Quando lo script viene eseguito, viene creata una cartella (quasi) vuota denominata `output`, poiché il processo di build non è ancora iniziato.

Modifichiamo la configurazione con `menuconfig`.

```
cd output
make menuconfig
```

Modificare queste configurazioni:

- Nome host di sistema in `lvgl-buildroot` in `system_configuration > System hostname`.
- Password di root in `lvgl-buildroot` in `system_configuration > Root password`.
- Abilitare SSH in `target-packages > Networking applications > openssh`

Creare l'immagine

```
make
```

Cosa è successo?

Esploriamo il contenuto della cartella di output e cosa contiene.

build

Questa cartella contiene i file intermedi e gli artefatti della build per i vari pacchetti in fase di compilazione. Ogni pacchetto ha una propria sottodirectory all'interno della cartella `build`, dove avviene il processo di compilazione. Include file come file di configurazione, codice sorgente e file oggetto generati durante il processo di compilazione.

host

La cartella `host` contiene file e binari compilati per il sistema host anziché per il sistema target. Include strumenti e utilità necessari per compilare i pacchetti o per eseguire il sistema di build stesso. Può contenere compilatori, strumenti di compilazione e librerie necessari per supportare il processo di compilazione per il sistema target.

images

Questa directory contiene le immagini di output finali generate per il sistema target, come immagini del file system, immagini del kernel o immagini del bootloader. A seconda della configurazione, si potrebbero trovare file come `rootfs.tar`, `zImage`, `uImage`, o altri, pronti per essere distribuiti sull'hardware target.

target

La cartella `target` contiene i file specificamente destinati al sistema di destinazione. Questo include il file system di root e qualsiasi file aggiuntivo che verrà incluso nell'ambiente di destinazione. La struttura all'interno di questa cartella spesso imita la struttura delle directory di un sistema Linux standard, con directory come `bin`, `lib`, `etc`, `usr` e altre, che contengono i file binari, le librerie, i file di configurazione e altri componenti necessari per il corretto funzionamento del sistema target.

Eseguire il flashing dell'immagine

Inserire la scheda SD nel laptop e controllarne il punto di montaggio. In genere è etichettata come sda o sdb, ma si può usare il comando lsblk per confermare questa informazione.

Se è montata su /dev/sda, si esegue il seguente comando:

```
sudo dd if=images/sdcard.img of=/dev/sda
```

Come indicato in *images*, l'immagine di output si trova in `images` e si chiama `sdcard.img`.

Collegare un cavo Ethernet al RPi4 e assicurarsi che il laptop e l'RPi4 siano sulla stessa rete.

Si può usare `ifconfig` o `ip a` per trovare il proprio indirizzo IP. Poi, si usa `nmap` o qualsiasi altro strumento per scansionare la rete e trovare l'indirizzo IP del RPi4.

```
nmap -sn <YOUR_IP_ADDRESS>/24 | grep lvgl
```

Tutto questo può essere fatto con l'UART invece che con SSH se non si vuol connettere l'RPi4 alla rete.

Applicazione LVGL

Questa sezione fornisce informazioni sui passaggi da seguire per ottenere un'applicazione personalizzata che utilizzi LVGL in esecuzione sulla scheda.

Aggiornare RootFS

A seconda dell'applicazione, potrebbe essere necessario aggiornare il rootfs. Prendiamo come esempio la compilazione di LVGL con DRM. Il sistema deve avere `libdrm` installato.

```
cd output
make menuconfig
```

Per cercare un pattern di stringhe nella configurazione, premere `/` seguito dal pattern desiderato. Ad esempio, cercare `libdrm`. Si dovrebbe trovare `BR2_PACKAGE_LIBDRM` impostato su **[=n]**. Il campo "Location" indica dove trovare questa opzione. Premendo il tasto numerico corrispondente (9-0), è possibile accedere direttamente all'opzione.

Nella finestra `Search Results`, la sezione `Depends on` elenca i pacchetti o le opzioni richiesti che devono essere abilitati (o disabilitati) per rendere configurabile il pacchetto target. Per qualsiasi pacchetto che si desidera aggiungere, queste dipendenze devono essere soddisfatte; in caso contrario, l'opzione rimarrà nascosta.

Una volta completata la configurazione del pacchetto, si compila l'ambiente per aggiungere i pacchetti (`make` nella cartella **output**).

Per verificare che la libreria sia stata installata, possiamo trovarla nella sysroot di destinazione:

```
find build/ -name "*libdrm*"
```

Si dovrebbero vedere la cartella "include" e i file .so.

Generare l'SDK e configurare l'ambiente

Generare un SDK utilizzabile per la cross-compilazione dell'applicazione per il target (RPi4).

```
make sdk
```

Viene generato un file `.tar.gz` in output/images. Questo è l'SDK!

Per utilizzarlo, estrarlo ovunque.

```
mkdir -p ~/sdk
tar -xzf images/aarch64-buildroot-linux-gnu_sdk-buildroot.tar.gz -C ~/sdk
cd ..
```

Per configurare l'ambiente

```
mkdir application && cd application
touch setup-build-env.sh && chmod +x setup-build-env.sh
```

Lo script `setup-build-env.sh` si presenta così:

```
#!/bin/bash

SDK_PATH="$HOME/sdk/aarch64-buildroot-linux-gnu_sdk-buildroot"

export PATH="${SDK_PATH}/bin:${PATH}"

export SYSROOT="${SDK_PATH}/aarch64-buildroot-linux-gnu/sysroot"

export CROSS_COMPILE="aarch64-buildroot-linux-gnu-"

export CC="${CROSS_COMPILE}gcc"
export CXX="${CROSS_COMPILE}g++"
export LD="${CROSS_COMPILE}ld"
export AR="${CROSS_COMPILE}ar"
export AS="${CROSS_COMPILE}as"

export CFLAGS="--sysroot=${SYSROOT}"
export LDFLAGS="--sysroot=${SYSROOT}"
```

Build dell'applicazione

L'ambiente è ora configurato e siamo pronti per compilare un'applicazione utilizzando il repository `lv_benchmark` ispirato a lv_port_linux.

Tornare alla radice del progetto e clonare il repository:

```
git clone --recurse-submodules https://github.com/EDGEMTech/lv_benchmark.
↪git
```

L'applicazione è configurata per essere eseguita su fbdev. È possibile mantenere la configurazione di default o modificarla in base alle proprie preferenze.

Compilare l'applicazione

```
cd lv_benchmark
cmake -B build -S . -DCMAKE_C_COMPILER=${CROSS_COMPILE}gcc -DCMAKE_CXX_
↪COMPILER=${CROSS_COMPILE}g++ -DCMAKE_SYSROOT=${SYSROOT} -DCMAKE_C_FLAGS=
↪"--sysroot=${SYSROOT}" -DCMAKE_CXX_FLAGS="--sysroot=${SYSROOT}"
make -j $(nproc) -C build
```

Verificare che l'eseguibile di output sia stato compilato con la toolchain corretta:

```
file bin/lvgl-app
cd ../..
```

L'output dovrebbe contenere queste informazioni:

- ARM aarch64
- interpreter /lib/ld-linux-aarch64.so.1

Impostare un overlay rootfs

In Buildroot, un overlay rootfs (o overlay del filesystem di root) è un meccanismo che consente di aggiungere file, directory e configurazioni personalizzate direttamente nel filesystem di root dell'immagine target durante il processo di build. È un modo per estendere o modificare il contenuto del filesystem di root senza alterare i pacchetti di base o ricompilare tutto.

Creare la struttura dell'overlay rootfs e includere l'eseguibile della $application compilata.

```
mkdir -p resources/board/rootfs_overlay/usr/bin
cp application/lv_benchmark/bin/lvgl-app resources/board/rootfs_overlay/
↪usr/bin/
```

Aggiornare la configurazione.

```
cd output
make menuconfig
```

Trovare l'opzione dell'overlay rootfs `System configuration > Root filesystem overlay directories` e utilizzare il path relativo all'overlay rootfs con l'applicazione (**../resources/board/rootfs_overlay**) e salvare la configurazione.

Ora utilizzare il comando `make` per applicare l'overlay rootfs e verificare che la sysroot sia stata aggiornata come previsto.

```
find . -name lvgl-app
```

Eseguire il flashing dell'immagine ed eseguire l'applicazione di benchmark sulla board.

4.9.2 FreeRTOS

TODO

4.9.3 MQX RTOS

Consultare la homepage di MQX RTOS

TODO

4.9.4 NuttX RTOS

Cos'è NuttX?

NuttX è un sistema operativo real-time (RTOS) maturo e sicuro, con particolare attenzione alla conformità agli standard tecnici e alle dimensioni ridotte. È scalabile da microcontrollori e microprocessori da 8 bit a 64 bit ed è conforme agli standard Portable Operating System Interface (POSIX) e American National Standards Institute (ANSI), nonché a molti sottosistemi simili a Linux. Il modo migliore per pensare a NuttX è considerarlo come un piccolo Unix/Linux per microcontrollori.

Punti di Forza di NuttX

- **Piccolo** - Si adatta e funziona su microcontrollori con soli 32 kB di Flash e 8 kB di RAM.
- **Conforme** - Si impegna a essere il più compatibile possibile con POSIX e Linux.
- **Versatile** - Supporta numerose architetture (ARM, ARM Thumb, AVR, MIPS, OpenRISC, RISC-V a 32 e 64 bit, RX65N, x86-64, Xtensa, Z80/Z180, ecc.).
- **Modulare** - Il suo design modulare consente agli sviluppatori di selezionare solo ciò che conta davvero e di utilizzare i moduli per includere nuove funzionalità.
- **Popolare** - NuttX è utilizzato da molte aziende in tutto il mondo. Probabilmente è già stato utilizzato un prodotto con NuttX senza sapere che eseguiva NuttX.
- **Prevedibile** - NuttX è un kernel "Realtime preemptible", quindi lo si può usare per creare applicazioni prevedibili per il controllo in tempo reale.

Perché NuttX + LVGL?

Sebbene NuttX abbia una propria libreria grafica chiamata NX, LVGL è una buona alternativa perché gli utenti potrebbero trovare demo più accattivanti e riutilizzare il codice di progetti precedenti. LVGL è una libreria GUI di alto livello Basata su Componenti Object-Oriented, che potrebbe adattarsi perfettamente a un RTOS con funzionalità avanzate come NuttX. LVGL è implementato in C e le sue API sono in C.

Ecco alcuni vantaggi dell'utilizzo di LVGL in NuttX

- Sviluppare prima la GUI in Linux e, una volta completata, compilarla per NuttX. Niente di più, nessuna perdita di tempo.
- Di solito, lo sviluppo di GUI per RTOS di basso livello richiede più iterazioni per ottenere il risultato desiderato, dove ogni iterazione consiste in **Modifica del codice > Build > Flash > Run**. Utilizzando LVGL, Linux e NuttX si può ridurre questo processo e testare tutto sul computer e, una volta completato, compilarlo su NuttX e il gioco è fatto.

NuttX + LVGL potrebbero essere utilizzati per

- Delle demo della GUI per dimostrare le capacità grafiche della scheda.
- Prototipazione rapida della GUI per la presentazione MVP (Minimum Viable Product).
- Visualizzazione dei dati dei sensori direttamente e facilmente sulla scheda senza utilizzare un computer.
- Prodotti finali con una GUI senza touchscreen (ad esempio, un'interfaccia per stampante 3D che utilizza un encoder rotativo per l'inserimento dei dati).
- Prodotti finali con touchscreen (e tutti i tipi di ammennicoli).

Come iniziare con NuttX e LVGL?

Ci sono molte board nella NuttX mainline con supporto per LVGL. Usiamo la STM32F429IDISCOVERY come esempio, perché è una scheda molto popolare.

Per prima cosa si devono installare i prerequisiti sul sistema.

Utilizziamo Windows Subsystem for Linux

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

Ora creiamo un'area di lavoro in cui salvare i nostri file.

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

Cloniamo i repository NuttX e Apps.es

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

Simulatore NuttX

Il simulatore NuttX consente di eseguire NuttX e LVGL sul PC. Questo può essere particolarmente utile per sviluppare interfacce utente LVGL con feedback visivo immediato e anche per sviluppare app NuttX indipendenti dalla piattaforma.

Configurare NuttX per utilizzare la board `sim` e LVGL Demo

```
$ ./tools/configure.sh sim:lvgl_fb
$ make
```

Se tutto è andato bene, si avrà il file `nuttx`

```
$ ls -l nuttx
-rwxr-xr-x 1 root root 2819920 May 12 15:01 nuttx
```

Esecuzione del Simulatore NuttX

```
$ ./nuttx
```

Utilizzando il terminale 'NSH>' avviare la demo LVGL:

```
nsh> lvgldemo
```

NuttX On-Device

Se non si ha una STM32F429I-DISCO, questa procedura sarà simile per altre schede.

Configurare NuttX per utilizzare la scheda stm32f429i-disco e la demo LVGL

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

Se tutto è andato a buon fine, si dovrebbe avere il file `nuttx.bin` da flashare sulla scheda:

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

Flashare il firmware sulla scheda usando OpenOCD

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

Resettare la scheda e, usando il terminale 'NSH>', avviare la demo LVGL:

```
nsh> lvgldemo
```

Utilizzo Personalizzato

L'app di esempio chiamata `lvgldemo` usa l'integrazione NuttX di LVGL per eseguire le demo in NuttX.

https://github.com/apache/nuttx-apps/tree/master/examples/lvgldemo

Viene utilizzata da tutte le configurazioni di LVGL. Vedere `sim:lvgl_fb` e `sim:lvgl_lcd`.

La si può usare come riferimento per aggiungere LVGL alla propria app NuttX. Iniziamo copiando il contenuto di `lvgldemo.c`. Gestisce i driver dei display basati su fb e lcd, il driver di input touchscreen e libuv.

Supporto Driver NuttX

- **fbdev** - LVGL può creare un display per i dispositivi `/dev/fb*`.
- **lcd** - LVGL può creare un display per i dispositivi `/dev/lcd*`.

- **input/uinput touchscreen** - LVGL può creare uno o più indev per dispositivi touch che utilizzano il sistema input/uinput NuttX.
- **libuv** - LVGL supporta l'utilizzo di libuv come "event loop manager". Le app libuv possono integrarsi perfettamente.

Non è ancora disponibile il supporto per le tastiere input/uinput. Il driver della tastiera sim NuttX X11 emette codici tasto non standard, quindi l'integrazione di LVGL con esso non sarebbe generica, se fosse aggiunto.

Altre integrazioni

- Integrazione del profiler LVGL+NuttX
- Heap opzionale della cache immagini dedicato

Configurazioni

La libreria LVGL è una "app" di NuttX. È possibile configurare LVGL in `menuconfig`.

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

Il file `Kconfig` di LVGL fornisce le opzioni di configurazione disponibili per il sistema di build di NuttX. Le configurazioni LVGL in `menuconfig` si trovano in **Application Configuration > Graphics Support > Light and Versatile Graphic Library (LVGL) > LVGL configuration**.

Per trovare la posizione di una configurazione specifica, premere il tasto "/" per aprire un'interfaccia di ricerca.

Ecco alcune configurazioni utilizzabili per personalizzare la configurazione di NuttX e LVGL:

- **LV_USE_NUTTX_INDEPENDENT_IMAGE_HEAP** - È possibile abilitare o disabilitare l'heap delle immagini LVGL in NuttX. Per default, è disabilitato. Se lo si abilita, LVGL utilizzerà invece l'heap NuttX.

Utilizzare *LV_STDLIB_CLIB* per *LV_USE_STDLIB_MALLOC*, *LV_USE_STDLIB_STRING* o *LV_USE_STDLIB_SPRINTF* è ragionevole, poiché NuttX implementa completamente queste API della libreria standard. La scelta di utilizzare `malloc` di NuttX dipende dal fatto che si desideri che LVGL allochi dall'heap globale di NuttX o utilizzi il proprio.

Dov'è `LV_OS_NUTTX`?

NuttX cerca di essere compatibile con POSIX ove possibile, il che significa che supporta i pthread (thread POSIX). Per abilitare le funzionalità del sistema operativo in LVGL su NuttX, impostare *LV_USE_OS* a *LV_OS_PTHREAD*. Il motivo principale per cui si abilitano le funzionalità del sistema operativo è il rendering multi-core. Vedere *LV_DRAW_SW_DRAW_UNIT_CNT*. Altrimenti non è necessario impostarlo.

Utilizzo di una Versione Specifica di LVGL

Ecco i passaggi per utilizzare l'ultima versione (o qualsiasi) di LVGL in NuttX.

Assicurarsi innanzitutto che gli artefatti di build siano stati cancellati. Questo rimuoverà il file LVGL e lo ZIP decompressi esistenti, se presenti.

```
$ make distclean
```

Ora si può usare `git clone` per l'ultima versione di LVGL nell'albero delle app di NuttX. Persisterà anche dopo `make distclean` perché il sistema di build non rimuoverà lvgl se si tratta di un repository Git.

```
$ cd /path/to/nuttxspace/apps/graphics/lvgl
$ git clone https://github.com/lvgl/lvgl.git
```

Infine, si deve copiare il contenuto di `apps/graphics/lvgl/lvgl/Kconfig` al centro di `apps/graphics/lvgl/Kconfig`. Vedere la sezione "PASTE THE CONTENTS ..." qui sotto.

`apps/graphics/lvgl/Kconfig`:

```
#
# For a description of the syntax of this configuration file,
# see the file kconfig-language.txt in the NuttX tools repository.
#

menuconfig GRAPHICS_LVGL
    bool "Light and Versatile Graphic Library (LVGL)"
    default n
    ---help---
            Enable support for the LVGL GUI library.

if GRAPHICS_LVGL

(PASTE THE CONTENTS OF apps/graphics/lvgl/lvgl/Kconfig HERE)

config LV_OPTLEVEL
    string "Customize compilation optimization level"
    default ""

endif # GRAPHICS_LVGL
```

Dove si possono trovare maggiori informazioni?

- Questo post del blog: LVGL on LPCXpresso54628
- Mailing list NuttX: Apache NuttX Mailing List

4.9.5 PX5 RTOS

Vedere la homepage di PX5 RTOS

TODO

4.9.6 QNX

Cos'è QNX?

QNX è un sistema operativo commerciale rilasciato per la prima volta nel 1980. Il sistema operativo si basa su un design a microkernel, con file system, stack di rete e vari altri driver, ognuno dei quali eseguito in un proprio processo con uno spazio di indirizzamento separato.

Per maggiori dettagli, consultare www.qnx.com.

Punti di forza di QNX

- Solo a 64 bit, funziona su x86_64 e ARMv8
- Richiede una MMU poiché la progettazione impone la separazione tra i processi
- Supporto per migliaia di processi e milioni di thread
- Fino a 64 core, fino a 16 TB di RAM
- Supporto per la virtualizzazione (come host e guest)
- Compatibilità POSIX completa
- Certificazione di sicurezza secondo vari standard automobilistici, industriali e medicali

Come eseguire LVGL su QNX?

Esistono due modi per utilizzare LVGL nei progetti QNX. Il primo è simile a come LVGL viene utilizzato su altri sistemi. Il secondo consiste nel compilare LVGL come libreria 'shared' o statica.

Includere LVGL nel Progetto

Seguire le istruzioni generiche per iniziare a usare LVGL. Dopo aver copiato `lv_conf_template.h` in `lv_conf.h` si apportano le seguenti modifiche a quest'ultimo:

1. Abilitare il supporto QNX:

```
#define LV_USE_QNX 1
```

2. Impostare la profondità di colore a 32:

```
#define LV_COLOR_DEPTH 32
```

3. (Facoltativo) Abilitare il doppio buffering:

```
#define LV_QNX_BUF_COUNT 2
```

Compilare LVGL come libreria

Notare che questo metodo è un'alternativa all'inclusione di LVGL nel progetto. Se si sceglie di compilare una libreria, non è necessario seguire le istruzioni della sezione precedente.

La directory di livello superiore `qnx` include un file make ricorsivo per compilare LVGL, sia come libreria condivisa che come libreria statica per le architetture supportate. Per compilare tutte le librerie, è sufficiente richiamare `make` in questa directory:

```
# cd $(LVGL_ROOT)/env_support/qnx
# make
```

Se si preferisce compilare per un'architettura e una variante specifiche, si va alla directory appropriata e si esegue `make` da lì. Ad esempio, per creare una libreria condivisa per ARMv8:

```
# cd $(LVGL_ROOT)/env_support/qnx/aarch64/so.le
# make
```

Come regola generale, se si desidera avere una sola applicazione LVGL nel sistema, è meglio utilizzare una libreria statica. Se ce n'è più di una, e soprattutto se vengono eseguite contemporaneamente, è meglio utilizzare la libreria condivisa.

Prima di creare la libreria, è consigliabile modificare `$(LVGL_ROOT)/env_support/qnx/lv_conf.h`, ad esempio per aggiungere font o disabilitare il doppio buffering.

Scrivere un'Applicazione LVGL

Per creare un'applicazione LVGL per QNX, seguire questi passaggi nel codice:

1. Inizializzare la libreria.
2. Creare una finestra.
3. Aggiungere i dispositivi di input.
4. Creare l'interfaccia utente.
5. Eseguire il "loop" di eventi.

I passaggi 2, 3 e 5 utilizzano chiamate specifiche per QNX, ma il resto del codice dovrebbe essere identico a quello di un'applicazione LVGL scritta per qualsiasi altra piattaforma.

Il codice seguente mostra come creare un'applicazione "Hello World":

```
#include <lvgl.h>

int
main(int argc, char **argv)
{
    /* Initialize the library. */
    lv_init();

    /* Create a 800x480 window. */
    lv_display_t *disp = lv_qnx_window_create(800, 480);
    lv_qnx_window_set_title(disp, "LVGL Example");

    /* Add keyboard and mouse devices. */
    lv_qnx_add_keyboard_device(disp);
    lv_qnx_add_pointer_device(disp);

    /* Generate the UI. */
    lv_obj_set_style_bg_color(lv_screen_active(), lv_color_hex(0x003a57),␣
↪LV_PART_MAIN);

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Hello world");
    lv_obj_set_style_text_color(lv_screen_active(), lv_color_hex(0xffffff),
↪ LV_PART_MAIN);
    lv_obj_align(label, LV_ALIGN_CENTER, 0, 0);

    /* Run the event loop until it exits. */
    return lv_qnx_event_loop(disp);
}
```

Building dell'Applicazione

Il building dell'applicazione consiste nel compilare il codice sorgente con gli header LVGL e poi nel linkarlo alla libreria. Questo può essere fatto in molti modi, utilizzando diversi sistemi di build. Di seguito è riportato un semplice file make per l'esempio precedente, che compila per ARMv8 con la libreria "shared":

```
CC=qcc -Vgcc_ntoaarch64le

LVGL_ROOT=$(HOME)/src/lvgl
CCFLAGS=-I$(LVGL_ROOT)/env_support/qnx -I$(LVGL_ROOT)
LDFLAGS=-lscreen -llvgl -L$(LVGL_ROOT)/env_support/qnx/aarch64/so.le

lvgl_example: lvgl_example.c
    $(CC) $(CCFLAGS) -Wall -o $@ $< $(LDFLAGS)

clean:
    rm -f *.o *~ lvgl_example
```

4.9.7 RT-Thread RTOS

Cos'è RT-Thread?

RT-Thread è un sistema operativo real-time (RTOS) open source, neutrale e basato sulla community. RT-Thread è disponibile in **versione Standard** e **versione Nano**. Per i sistemi a microcontrollore (MCU) con risorse limitate, la versione Nano, che richiede solo 3 KB di memoria Flash e 1,2 KB di RAM, può essere personalizzata con strumenti di facile utilizzo. Per i dispositivi IoT ricchi di risorse, RT-Thread può utilizzare lo strumento di gestione dei **pacchetti**

software online, insieme agli strumenti di configurazione del sistema, per ottenere un taglio modulare rapido e intuitivo, importare senza problemi pacchetti software complessi e ottenere così funzioni complesse come l'interfaccia grafica di Android e gli effetti di scorrimento touch, gli effetti di interazione vocale intelligente e così via.

Caratteristiche Principali

- Progettato per dispositivi con risorse limitate, il kernel minimo richiede solo 1,2 KB di RAM e 3 KB di Flash.
- Una varietà di interfacce standard, come POSIX, CMSIS, ambiente applicativo C++.
- Ha componenti ricchi e un ecosistema di pacchetti prospero e in rapida crescita
- Stile di codice elegante, facile da usare, leggere e padroneggiare.
- Elevata scalabilità. RT-Thread ha un'architettura software scalabile di alta qualità, "loose coupling" [accoppiamento libero], modularità, è facile da personalizzare ed espandere.
- Supporta applicazioni ad alte prestazioni.
- Supporta tutti i principali strumenti di compilazione come GCC, Keil e IAR.
- Supporta un'ampia gamma di architetture e chip

Come eseguire LVGL su RT-Thread?

[documentazione cinese]

LVGL è registrato come softwarepackage di RT-Thread. Utilizzando Env tool o RT-Thread Studio IDE, gli utenti di RT-Thread possono scaricare facilmente il codice sorgente di LVGL e integrarlo nel progetto RT-Thread.

La community RT-Thread ha portato LVGL su diversi BSP [Board Support Package]:

BSP	BSP
Simulatore QEMU	Infineon psoc6-evaluationkit-062S2
Simulatore di Visual Studio	Renesas ra6m3-ek
Nuvoton numaker-iot-m487	Renesas ra6m4-cpk
Nuvoton numaker-pfm-m487	Renesas ra6m3-hmi
Nuvoton nk-980iot	STM32H750 ART-Pi
Nuvoton numaker-m2354	STM32F469 Discovery
Nuvoton nk-n9h30	STM32F407 explorer
Nuvoton numaker-m032ki	STM32L475 pandora
Nuvoton numaker-hmi-ma35d1	NXP imxrt1060-evk
Nuvoton numaker-iot-m467	Raspberry PICO
Nuvoton numaker-m467hj	NXP LPC55S69
synwit swm341	

Tutorial

- Introduzione a RT-Thread e a come eseguire LVGL su RT-Thread nei simulatori
- Come importare un progetto BSP con il codice più recente in RT-Thread Studio
- Come usare LVGL con RT-Thread Studio nella board STM32F469 Discovery
- Canale YouTube di RT-Thread
- Centro di documentazione di RT-Thread

4.9.8 Torizon OS

Torizon OS è una piattaforma software open source che mira a semplificare lo sviluppo e la manutenzione di software Linux embedded.

Utilizzando i container Docker, consente agli sviluppatori di concentrarsi sull'applicazione anziché creare e gestire una distribuzione personalizzata.

Questa guida spiega come creare un'immagine Docker contenente LVGL e una semplice applicazione demo che può essere distribuita su qualsiasi dispositivo Toradex con TorizonOS.

Prerequisiti

Per seguire questa guida è ovviamente necessario disporre di un SoM Toradex e di una board carrier. Ulteriori informazioni sono disponibili sul sito web di Toradex.

> **Nota**
>
> È necessario avere familiarità con alcuni principi fondamentali di Docker per poter comprendere e seguire questa guida. Inoltre, Docker deve essere presente sull'host di sviluppo.

La documentazione di Toradex è una risorsa utile. Questo articolo contiene molti riferimenti ad essa.

Configurazione della scheda

Si inizia installando TorizonOS utilizzando Toradex Easy Installer.. Seguire le guide ufficiali di avvio.

Nell'Easy Installer, selezionare TorizonCore come sistema operativo da installare.

Copiare l'indirizzo IP assegnato al dispositivo dal server DHCP sulla propria rete. L'indirizzo viene visualizzato nell'angolo in basso a destra. Verrà utilizzato in seguito.

Una volta completata la configurazione, il dispositivo si avvierà in TorizonOS.

Creazione dell'immagine Docker

Toradex fornisce un'estensione VSCode che offre una raccolta di template utilizzati per configurare e automatizzare le attività necessarie per la cross-compilazione delle applicazioni e la creazione di immagini Docker.

Tuttavia, questa guida spiega come eseguire manualmente tali operazioni.

> **Nota**
>
> Il template per un'applicazione LVGL è attualmente in fase di aggiunta all'estensione VSCode e sarà disponibile a breve.

Per creare un container Torizon, è necessario abilitare l'emulazione Docker ARM digitando i seguenti comandi:

```
docker run --rm -it --privileged torizon/binfmt
docker run --rm -it --pull always --platform linux/arm64/v8 debian arch
```

Il secondo comando dovrebbe stampare `aarch64`.

Ulteriori informazioni sull'ambiente di build per i container Torizon sono disponibili qui.

Iniziamo creando una directory che conterrà il `Dockerfile` e il codice sorgente che verrà utilizzato per creare l'immagine.

Digitare i seguenti comandi in una finestra di terminale:

```
mkdir -p ~/lvgl_torizon_os/
cd ~/lvgl_torizon_os/
touch Dockerfile

git clone --depth 1 https://github.com/lvgl/lv_port_linux
git -C lv_port_linux submodule update --init
```

Questi comandi creano la directory del progetto e il `Dockerfile`. `git` viene utilizzato per scaricare i repository `lv_port_linux` e `lvgl` da Github.

> **Nota**
>
> Per default, `lv_port_linux` è configurato per utilizzare il dispositivo framebuffer legacy `/dev/fb0`. È anche possibile utilizzare un altro backend di rendering abilitando le opzioni corrette in `lv_port_linux/lv_conf.h`.

Ora si modifica il Dockerfile. Copiare e incollare il blocco seguente nel file:

```
ARG CROSS_SDK_BASE_TAG=3.2.1-bookworm
ARG BASE_VERSION=3.2.1-bookworm
##
# Board architecture
# arm or arm64
##
ARG IMAGE_ARCH=arm64

##
# Directory of the application inside container
##
ARG APP_ROOT=/usr/lvgl_application

# BUILD ---------------------------------------------------------------
↪-----
FROM torizon/debian-cross-toolchain-${IMAGE_ARCH}:${CROSS_SDK_BASE_TAG} AS␣
↪build

ARG APP_ROOT
ARG IMAGE_ARCH

RUN apt-get -q -y update && \
    apt-get -q -y install && \
    apt-get clean && apt-get autoremove && \
    apt-get install -q -y curl git cmake file && \
    rm -rf /var/lib/apt/lists/*

COPY . ${APP_ROOT}
WORKDIR ${APP_ROOT}

# Compile lv_port_linux
RUN CC=aarch64-linux-gnu-gcc cmake -S ./lv_port_linux -B build
RUN make -j 4 -C ${APP_ROOT}/build

# DEPLOY --------------------------------------------------------------
↪-----
FROM --platform=linux/${IMAGE_ARCH} torizon/debian:${BASE_VERSION} AS␣
↪deploy

ARG IMAGE_ARCH
ARG APP_ROOT

RUN apt-get -y update && apt-get install -y --no-install-recommends \
&& apt-get clean && apt-get autoremove && rm -rf /var/lib/apt/lists/*
```

(continues on next page)

(continua dalla pagina precedente)

```
# Copy the lvglsim executable compiled in the build step to the $APP_ROOT␣
↪directory
# path inside the container
COPY --from=build ${APP_ROOT}/lv_port_linux/bin/lvglsim ${APP_ROOT}

# Command executed during runtime when the container starts
ENTRYPOINT [ "./lvglsim" ]
```

Il `Dockerfile` funge da ricetta per creare due immagini `build` e `deploy`.

Per prima cosa scaricare i pacchetti necessari per compilare il simulatore utilizzando il gestore di pacchetti Debian `apt-get`.

Dopo la compilazione, l'eseguibile risultante viene scritto in `lv_port_linux/bin/lvglsim`.

L'immagine `deploy` verrà distribuita sul dispositivo. L'eseguibile creato nell'immagine precedente viene copiato nella directory `/usr/bin` dell'immagine corrente.

Questo crea un'immagine più piccola che non include la tool chain e le dipendenze di building.

Le immagini vengono compilate con il seguente comando:

```
docker build . -t lvgl_app
```

Docker interpreterà il `Dockerfile` presente nella directory di lavoro corrente. L'argomento `-t` assegna un nome all'immagine risultante.

Al termine, assicurarsi che l'immagine sia elencata da Docker:

```
docker image list | grep lvgl_app
```

Dovrebbe visualizzare l'immagine insieme al suo ID che verrà utilizzato in seguito.

Deploy dell'immagine del container sul dispositivo

L'immagine è ora pronta per essere distribuita sul dispositivo. Esistono diversi modi per eseguire questa operazione.

Leggere questo articolo per maggiori informazioni.

In questa guida, configureremo un "container registry Docker" sull'host di sviluppo, accessibile da qualsiasi dispositivo della LAN. La scheda Toradex, essendo sulla stessa rete, sarà in grado di estrarre l'immagine dal registry.

Il registry viene creato come segue:

```
docker run -d -p 5000:5000 --name registry registry:2.7
```

Il flag `-d` esegue il container in modalità "detached". L'argomento `-p` specifica il mapping delle porte. Il "registry container" ascolterà sulla porta `TCP/5000` e verrà mappato esternamente alla stessa porta.

Eseguire il push dell'immagine creata nel passaggio precedente nel registry appena creato:

```
docker tag <IMAGE_ID> 127.0.0.1:5000/lvgl-app
```

```
docker push 127.0.0.1:5000/lvgl-app
```

Per default un "container registry" locale utilizza HTTP in chiaro, quindi l'istanza Docker in esecuzione sul dispositivo deve essere configurata per consentire il recupero di immagini da un repository 'non sicuro'.

Ottenere l'indirizzo IP dell'host di sviluppo e aprire una shell remota sul dispositivo.

```
sudo su # When prompted type in the password of the torizon user

# Be sure to replace set the IP address of your host instead
cat << heredoc > /etc/docker/daemon.json
{
     "insecure-registries" : ["<IP_ADDR_OF_DEVELOPMENT_HOST>:5000"]
```

(continues on next page)

(continua dalla pagina precedente)

```
}
heredoc

systemctl restart docker.service

# Return to regular user
exit

# Pull the image from your development host
docker pull <IP_ADDR_OF_DEVELOPMENT_HOST>:5000/lvgl-app
```

Per default, i container Docker sono isolati dall'hardware del sistema host. Sebbene questo sia ottimo per la sicurezza, un'applicazione embedded avrà ovviamente bisogno di accesso all'hardware.

Il container che esegue l'applicazione LVGL necessita di accesso al framebuffer `/dev/fb0` del dispositivo.

Utilizzando l'argomento `--device` è possibile mappare un dispositivo a un container.

Avviare il container in questo modo:

```
docker run --device /dev/fb0:/dev/fb0 <IMAGE_ID>
```

Conclusione

Ora si ha un'applicazione LVGL in esecuzione. Come procedere?

- Ora si è pronti per compilare l'applicazione LVGL per Torizon OS. Si consiglia di familiarizzare con l'estensione IDE VSCode, poiché semplificherà il flusso di lavoro.

  VSCode, è il modo migliore per sviluppare per Torizon OS. Se si usa un altro editor o IDE, si può sempre scrivere script per automatizzare le operazioni di compilazione/pushing/pulling.
- Leggere questo articolo er capire come progettare applicazioni per Torizon OS.
- Si consiglia inoltre di familiarizzare con Torizon Cloud e Torizon OS Builder quando ci si avvicina alla produzione.

4.9.9 Yocto

The Yocto Project (YP) è un progetto collaborativo open source che aiuta gli sviluppatori a creare sistemi Linux personalizzati, indipendentemente dall'architettura hardware.

Il progetto fornisce un set flessibile di strumenti e uno spazio in cui gli sviluppatori embedded di tutto il mondo possono condividere tecnologie, stack software, configurazioni e best practice, utilizzabili per creare immagini Linux personalizzate per dispositivi embedded e IoT, o ovunque sia necessario un sistema operativo Linux personalizzato.

L'obiettivo di questa sezione è facilitare la comprensione dei concetti di base di Yocto e aiutare i principianti a iniziare a utilizzarlo.

Componenti principali del progetto Yocto

L'esecutore di task BitBake, insieme a vari tipi di file di configurazione, costituisce l'OpenEmbedded-Core (OE-Core). Questa sezione fornisce una panoramica di questi componenti, descrivendone l'utilizzo e le interazioni.

BitBake gestisce l'analisi e l'esecuzione dei file di dati. I dati stessi sono di vario tipo:

- *Ricette:* Forniscono dettagli su particolari componenti software.
- *Dati di Classi:* Estraggono informazioni di build comuni (ad esempio, come compilare un kernel Linux).
- *Dati di Configurazione:* Definiscono impostazioni specifiche della macchina, decisioni sulle policy e così via. I dati di configurazione fungono da collante per collegare tutto

BitBake sa come combinare più sorgenti di dati e fa riferimento a ciascuna sorgente dati come a un layer.

Ecco alcuni brevi dettagli su questi componenti principali.

BitBake

BitBake è lo strumento al centro del sistema di compilazione OpenEmbedded ed è responsabile dell'analisi dei metadati, della generazione di un elenco di task da essi e della loro esecuzione.

Questa sezione introduce brevemente BitBake. Per maggiori informazioni su BitBake, consultare il Manuale Utente di BitBake.

Per visualizzare un elenco delle opzioni supportate da BitBake, utilizzare uno dei seguenti comandi:

```
$ bitbake -h
$ bitbake --help
```

L'uso più comune di BitBake è `bitbake recipename`, dove `recipename` è il nome della ricetta che si desidera compilare (definitocome "target"). Il target spesso corrisponde alla prima parte del nome file di una ricetta (ad esempio, "foo" per una ricetta denominata `foo_1.3.0-r0.bb`). Quindi, per elaborare il file della ricetta `matchbox-desktop_1.2.3.bb`, si potrebbe digitare quanto segue:

```
$ bitbake matchbox-desktop
```

Potrebbero esistere diverse versioni di `matchbox-desktop`. BitBake sceglie quella selezionata dalla configurazione della distribuzione. Per maggiori dettagli su come BitBake sceglie tra diverse versioni target e provider, consultare la sezione "Preferences" del Manuale Utente di BitBake.

BitBake tenta inoltre di eseguire prima eventuali attività dipendenti. Quindi, ad esempio, prima di compilare `matchbox-desktop`, BitBake compilerebbe un cross-compilatore e `glibc` se non fossero già stati compilati.

Un'opzione BitBake utile da considerare è `-k` o `--continue`. Questa opzione indica a BitBake di provare a continuare l'elaborazione del job il più a lungo possibile anche dopo aver riscontrato un errore. Quando si verifica un errore, il target che ha generato l'errore e quelli che dipendono da esso non possono essere ripristinati. Tuttavia, utilizzando questa opzione, è comunque possibile elaborare altre dipendenze.

Ricette

I file con suffisso `.bb` sono file "ricette" (recipe). In generale, una ricetta contiene informazioni su un singolo software. Queste informazioni includono la posizione da cui scaricare il codice sorgente non modificato, eventuali patch da applicare a tale codice sorgente (se necessario), quali opzioni di configurazione speciali applicare, come compilare i file sorgente e come pacchettizzare l'output compilato.

Il termine "package" [pacchetto] viene talvolta utilizzato per riferirsi alle ricette. Tuttavia, poiché il termine "package" viene utilizzato per l'output pacchettizzato dal sistema di build di OpenEmbedded (ad esempio, i file `.ipk` o `.deb`), questo documento evita di utilizzare il termine "package" quando si fa riferimento alle ricette.

Classi

I file di classe (`.bbclass`) contengono informazioni utili da condividere tra i file di ricette. Un esempio è la classe autotools*, che contiene impostazioni comuni per qualsiasi applicazione compilata con `GNU Autotools <https://en.wikipedia.org/wiki/GNU_Autotools>`. Il capitolo "Classi" nel Manuale di Riferimento del Progetto Yocto fornisce dettagli sulle classi e su come utilizzarle.

Configurazioni

I file di configurazione (`.conf`) definiscono diverse variabili di configurazione che governano il processo di compilazione di OpenEmbedded. Questi file rientrano in diverse aree che definiscono le opzioni di configurazione della macchina, le opzioni di configurazione della distribuzione, le opzioni di ottimizzazione del compilatore, le opzioni di configurazione comuni generali e le opzioni di configurazione utente in `conf/local.conf`, che si trova nella Build Directory.

I Layer

I layer sono repository che contengono metadati correlati (ovvero set di istruzioni) che indicano al sistema di build di OpenEmbedded come compilare un target. Il modello a layer del progetto Yocto facilita la collaborazione, la condivisione, la personalizzazione e il riutilizzo all'interno dell'ambiente di sviluppo Yocto Project. I layer separano logicamente le informazioni per il progetto. Ad esempio, è possibile utilizzare un layer per contenere tutte le configurazioni di un particolare componente hardware. Isolare le configurazioni specifiche dell'hardware consente di condividere altri metadati utilizzando un layer diverso, in cui tali metadati potrebbero essere comuni a diversi componenti hardware.

Nell'ambiente di sviluppo Yocto Project sono presenti molti layer. Sia Yocto Project Compatible Layer Index che OpenEmbedded Layer Index contengono layer che è possibile utilizzare o sfruttare.

Per convenzione, i layer nel Progetto Yocto seguono una forma specifica. Conformarsi a una struttura nota consente a BitBake di fare ipotesi durante le build su dove trovare i tipi di metadati. È possibile trovare procedure e informazioni sugli strumenti (ad esempio `bitbake-layers`) per la creazione di layer adatti al Progetto Yocto nella sezione "comprensione e creazione di layer" dello Yocto Project Development Tasks Manual.

LVGL in Yocto

Questo capitolo serve come guida per aiutare a creare una ricetta per un'applicazione che utilizza LVGL. Sebbene il processo verrà illustrato per Raspberry Pi 3 (64 bit), può essere applicato a qualsiasi board in modo simile.

Pacchetti Host di Build

È necessario installare i pacchetti host essenziali sul proprio host di build. Il seguente comando installa i pacchetti host basati su una distribuzione Ubuntu

```
sudo apt install gawk wget git diffstat unzip texinfo gcc build-essential \
chrpath socat cpio python3 python3-pip python3-pexpect xz-utils \
debianutils iputils-ping python3-git python3-jinja2 python3-subunit zstd \
liblz4-tool file locales libacl1
```

> **Nota**
>
> Per i requisiti dei pacchetti host su tutte le distribuzioni Linux supportate, consultare la sezione Required Packages for the Build Host nello Yocto Project Reference Manual.

Utilizzare Git per clonare i repository richiesti

Dopo questa sezione, l'albero delle cartelle sarà simile a questo

```
lvgl_yocto_guide/
├── build/
│   ├── cache/
│   ├── conf/
│   ├── downloads/
│   └── ...
└── sources/
    ├── meta-openembedded
    ├── meta-raspberrypi
    └── poky
```

Iniziare la creazione dell'architettura delle cartelle

```
mkdir -p lvgl_yocto_guide/sources
```

Clonaare Poky, meta-openembedded e meta-raspberrypi nei sorgenti

```
cd lvgl_yocto_guide/sources
git clone --branch scarthgap https://git.yoctoproject.org/poky.git
git clone --branch scarthgap https://git.openembedded.org/meta-openembedded
git clone --branch scarthgap git://git.yoctoproject.org/meta-raspberrypi
```

Creare un'immagine di base

Per capire meglio cosa sta succedendo, creiamo l'immagine come se fosse stata preparata per noi

```
cd ../ # go back to the root folder
source sources/poky/oe-init-build-env
```

Ora si dovrebbe avere una cartella chiamata `build` accanto a `sources`. L'ultimo comando imposta anche la directory corrente sulla directory di build.

Nella directory di build, c'è una cartella `conf` con alcuni file al suo interno

```
└── build/
    └── conf/
        ├── bblayers.conf
        ├── conf-notes.txt
        ├── conf-summary.txt
        ├── local.conf
        └── templateconf.cfg
```

I file importanti per noi sono `local.conf` e `bblayers.conf`

Per aggiungere layer al progetto ci sono 2 opzioni:

Comandi Bitbake (Opzione 1)

Si deve stare nello stesso terminale in cui è stato eseguito il comando `source`. Si aggiungolo i layer col comando Bitbake in questo modo:

```
bitbake-layers add-layer ../sources/meta-openembedded
bitbake-layers add-layer ../sources/meta-raspberrypi
```

Modifica del file di configurazione (Opzione 2)

Aprire il file `conf/bblayers.conf` e aggiungere manualmente i path:

```
# POKY_BBLAYERS_CONF_VERSION is increased each time build/conf/bblayers.
↪conf
# changes incompatibly
POKY_BBLAYERS_CONF_VERSION = "2"

BBPATH = "${TOPDIR}"
BBFILES ?= ""

BBLAYERS ?= " \
${TOPDIR}/../sources/poky/meta \
${TOPDIR}/../sources/poky/meta-poky \
${TOPDIR}/../sources/poky/meta-yocto-bsp \
${TOPDIR}/../sources/meta-raspberrypi \
${TOPDIR}/../sources/meta-openembedded/meta-oe \
${TOPDIR}/../sources/meta-openembedded/meta-multimedia \
${TOPDIR}/../sources/meta-openembedded/meta-networking \
```

(continues on next page)

(continua dalla pagina precedente)

```
${TOPDIR}/../sources/meta-openembedded/meta-python \
"
```

Per assicurarsi che i layer siano stati aggiunti come previsto, eseguire il comando Bitbake per visualizzarli tutti:

```
bitbake-layers show-layers
```

Dovrebbero essere elencati i seguenti layer:

- core
- yocto
- yoctobsp
- raspberrypi
- openembedded-layer
- multimedia-layer
- networking-layer
- meta-python

Build per RaspberryPi3 64

Le configurazioni macchina disponibili per RaspberryPi possono essere elencate in questo modo:

```
ls ../sources/meta-raspberrypi/conf/machine/*.conf
```

Per compilare un'immagine per RaspberryPi3 64 bit, modificare il file `local.conf` sostituendo il valore di default `MACHINE ??=` in questo modo

```
MACHINE ??= "raspberrypi3-64"
```

Per compilare l'immagine da utilizzare, è necessario aggiungere anche quanto segue al file:

```
LICENSE_FLAGS_ACCEPTED = "synaptics-killswitch"
```

> **Nota**
>
> Ulteriori informazioni su questa variabile sono disponibili qui.

Tutto è configurato, è ora di compilare l'immagine:

```
bitbake core-image-base
```

Panoramica della build

Esaminiamo le cartelle di build per capire cosa è successo.

Downloads

La posizione della cartella può essere modificata con la variabile `DL_DIR`.

La cartella Downloads è una parte fondamentale dell'ambiente di build di Yocto, in quanto funge da cache locale per tutte le sorgenti e le dipendenze esterne. Questo meccanismo di caching è essenziale per build efficienti, in quanto garantisce che build ripetute non comportino perdite di tempo nel download ripetuto degli stessi file.

tmp/deploy

Questa cartella contiene gli artefatti di output finali pronti per il "deployment" [distribuzione]. Questi artefatti includono immagini del kernel, file system di root, pacchetti, SDK, file del bootloader e altri componenti generati durante il processo di build di Yocto.

Ci sono alcune cartelle chiave:

- **images**: contiene le immagini che possono essere flashate o distribuite sul dispositivo target. File come il kernel Linux, il file system di root (ad esempio, . ext4, .tar.gz, .squashfs), i bootloader (ad esempio, U-Boot) e altre immagini "bootabili" per il dispositivo si trovano qui. È organizzato in base alla macchina (o board target) per cui è stata compilata l'immagine.
- **rmp/deb/ipk**: Queste cartelle contengono i singoli pacchetti software generati durante la compilazione, nel formato di pacchetto specificato (RPM, DEB o IPK). Questi pacchetti vengono in genere creati durante la compilazione del progetto Yocto con il supporto per la gestione dei pacchetti abilitato. Possono essere successivamente installati sul dispositivo di destinazione utilizzando gli strumenti di gestione dei pacchetti.
- **sdk**: Questa sottodirectory contiene i Software Development Kit (SDK) che è possibile utilizzare per la cross-compilazione delle applicazioni per il sistema di target. Gli SDK vengono generati quando si utilizza il comando `bitbake -c populate_sdk`.

tmp/sysroots-components

Questa cartella rappresenta un approccio modulare introdotto in Yocto per la gestione delle sysroot. Suddivide la sysroot in singoli componenti, che possono essere considerati come blocchi costitutivi o pacchetti che costituiscono l'intero ambiente sysroot.

Ogni componente corrisponde a un pacchetto o dipendenza specifica che viene inserita nella sysroot. Questo approccio consente un riutilizzo più efficiente delle sysroot su più pacchetti o build e facilita il monitoraggio e la gestione delle dipendenze durante il processo di build.

La directory sysroot-components aiuta a ottimizzare il processo di build perché, se un pacchetto cambia, Yocto deve solo aggiornare il componente corrispondente anziché ricostruire o copiare l'intera sysroot.

Se sono stati seguiti i passaggi precedenti, ecco le cartelle che si troveranno:

- `all`: File indipendenti dall'architettura.
- `cortexa53`: File per l'architettura Cortex-A53 (ARMv8-A).
- `manifests`: File "track" installati nella sysroot per pacchetto.
- `raspberrypi3_64`: File specifici per Raspberry Pi 3 (64 bit).
- `x86_64`: File per l'architettura x86_64 (PC), in genere per tool di cross-compilazione.
- `x86_64-nativesdk`: File relativi all'SDK per la cross-compilazione su un host x86_64.

Ogni cartella corrisponde a componenti rilevanti per l'architettura specifica e, nel loro insieme, formano l'ambiente completo necessario per compilare ed eseguire il software per i sistemi target e host.

tmp/sysroots-uninative

La directory sysroots-uninative in Yocto viene utilizzata per supportare la funzionalità "uninative", che consente build più riproducibili garantendo che l'ambiente di build rimanga coerente su diversi sistemi host. In sostanza, fornisce un modo per utilizzare gli stessi strumenti di build nativi su diverse distribuzioni Linux.

tmp/work

La cartella `work` in Yocto è una directory chiave nella cartella `tmp` che contiene tutti gli artefatti di build temporanei per ogni pacchetto durante il processo di build. È qui che avviene la build e la compilazione effettiva dei singoli pacchetti o ricette. Ogni pacchetto (o "recipe" [ricetta]) elaborato da BitBake genera file e directory temporanei all'interno di questa cartella `work`.

La cartella `work` è in genere strutturata in base all'architettura della macchina e al pacchetto. Ecco come funziona in genere:

```
work/
└── <architecture>/
    └── <package>/
        └── <version>/
            ├── temp/
            ├── work/
            ├── sysroot-destdir/
            └── image/
```

Questa cartella è molto importante e di grande aiuto durante le fasi di debug. La sottocartella `temp` contiene log importanti che possono aiutare a diagnosticare i problemi di build. È qui che si andrà a cercare quando una build non riesce a capire cosa è andato storto.

Ricetta LVGL

Maggiori informazioni sulle ricette sono disponibili nella sezione *Ricette*.

C'è una ricetta in `meta-openembedded` fin dalla versione `honister`.

Tabella 3: Versione della ricetta lvgl

Branch	Ricetta
scarthgap (Yocto Project 5.0)	lvgl 9.1.0
nanbield (Yocto Project 4.3)	lvgl 8.3.10
mickledore (Yocto Project 4.2)	lvgl 8.1.0
langdale (Yocto Project 4.1)	lvgl 8.1.0
langdale (Yocto Project 4.1)	lvgl 8.1.0
kirkstone (Yocto Project 4.0)	lvgl 8.0.3

In questa guida, ci troviamo sul branch `scarthgap`, quindi utilizziamo lvgl 9.1.0.

Approfondiamo questa ricetta per capire cosa viene fatto. L'obiettivo è aggiungere questa libreria come oggetto "shared" nel rootfs target e generare un SDK con lvgl.

Questo è il path delle ricette lvgl: `lvgl_yocto_guide/sources/meta-openembedded/ meta-oe/recipes-graphics/lvgl`

Ecco l'architettura della cartella delle ricette lvgl:

```
lvgl
├── files
│   ├── 0002-fix-sdl-handle-both-LV_IMAGE_SRC_FILE-and-LV_IMAGE_S.patch
│   ├── 0003-Make-fbdev-device-node-runtime-configurable-via-envi.patch
│   ├── 0004-Factor-out-fbdev-initialization-code.patch
│   ├── 0005-Add-DRM-KMS-example-support.patch
│   ├── 0006-Add-SDL2-example-support.patch
│   ├── 0007-fix-cmake-generate-versioned-shared-libraries.patch
│   └── 0008-fix-fbdev-set-resolution-prior-to-buffer.patch
├── lv-conf.inc
├── lvgl_9.1.0.bb
└── lvgl-demo-fb_9.1.0.bb
```

- La cartella `file` contiene tutte le patch che possono essere applicate durante la compilazione della ricetta.
- `lv_conf.inc` è un file include, che di solito contiene impostazioni di configurazione comuni per LVGL che possono essere condivise tra più ricette.

- `lvgl_9.1.0.bb` è la ricetta per compilare la libreria lvgl.
- `lvgl-demo-fb_9.1.0.bb` è una ricetta per compilare un'applicazione utilizzando lvgl.

Per ora, comprendiamo la ricetta della libreria lvgl.

```
# SPDX-FileCopyrightText: Huawei Inc.
#
# SPDX-License-Identifier: MIT

HOMEPAGE = "https://lvgl.io/"
DESCRIPTION = "LVGL is an OSS graphics library to create embedded GUI"
SUMMARY = "Light and Versatile Graphics Library"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://LICENCE.txt;md5=bf1198c89ae87f043108cea62460b03a
 ↪"

SRC_URI = "\
    git://github.com/lvgl/lvgl;protocol=https;branch=master \
    file://0002-fix-sdl-handle-both-LV_IMAGE_SRC_FILE-and-LV_IMAGE_S.patch␣
 ↪\
    file://0007-fix-cmake-generate-versioned-shared-libraries.patch \
    file://0008-fix-fbdev-set-resolution-prior-to-buffer.patch \
    "
SRCREV = "e1c0b21b2723d391b885de4b2ee5cc997eccca91"

inherit cmake

EXTRA_OECMAKE = "-DLIB_INSTALL_DIR=${baselib} -DBUILD_SHARED_LIBS=ON"
S = "${WORKDIR}/git"

require lv-conf.inc

do_install:append() {
    install -d "${D}${includedir}/${PN}"
    install -m 0644 "${S}/lv_conf.h" "${D}${includedir}/${PN}/lv_conf.h"
}

FILES:${PN}-dev += "\
    ${includedir}/${PN}/ \
    "
```

LICENSE: Indica la licenza del software, indicando che è distribuito con licenza MIT.

LIC_FILES_CHKSUM: Questa variabile contiene un checksum (in questo caso, un hash MD5) per il file di licenza (qui si presume sia LICENCE.txt). Questo aiuta a garantire l'integrità del file di licenza incluso nel pacchetto. Nel repository LVGL, è presente un file LICENSE.txt. Per ottenere il valore dell'hash del file, è possibile eseguire questo comando: `md5sum LICENSE.txt`

SRC_URI: Specifica le posizioni del codice sorgente e delle patch per la ricetta:

- Il repository sorgente principale per LVGL che specifica il branch master.
- I seguenti argomenti sono i file di patch locali che verranno applicati al codice sorgente durante il processo di build.

SRCREV: Definisce il commit specifico (in questo caso, un hash Git SHA) da cui verrà recuperato il codice sorgente. Questo garantisce che la build utilizzi una versione coerente del codice.

inherit: Questa riga indica che la ricetta utilizza la classe `cmake`, che fornisce funzionalità per la creazione di progetti che utilizzano CMake come sistema di build.

EXTRA_OECMAKE: Opzioni aggiuntive passate a CMake durante la fase di configurazione.

S: Questa variabile definisce la directory sorgente in cui verrà posizionato il codice sorgente decompresso dopo il recupero. `${WORKDIR}` è una variabile standard in BitBake che punta alla directory di lavoro per la ricetta. require: Questa riga include un altro file di configurazione, `lv-conf.inc`, che probabilmente contiene opzioni di configurazione aggiuntive o variabili specifiche della libreria LVGL.

FILES: Questa è una variabile BitBake utilizzata per specificare i file che devono essere inclusi in un particolare pacchetto. In questo caso, la variabile specifica i file per un pacchetto correlato allo sviluppo (ad esempio, file header).

I Task della Ricetta

Quando una ricetta viene compilata, eseguirà più task. È possibile eseguire ogni task manualmente per capire cosa viene generato in ogni passaggio, oppure è possibile eseguire `bitbake lvgl` per eseguire tutti i task.

Fetch (do_fetch)

```
bitbake lvgl -c fetch
```

Il task "fetch" recupera il codice sorgente del pacchetto dal repository locale o remoto.

L'indirizzo del Repository di recupero dev'essere memorizzato nella variabile **SRC_URI**. Nella variabile **SRCREV** viene definito l'hash di commit del repository GitHub.

Una volta completato il task di recupero (fetch), è possibile trovare i codici sorgente recuperati in `build/downloads`.

Per questa ricetta, si troverà una nuova cartella qui: `lvgl_yocto_guide build/downloads/git2/github.com.lvgl.lvgl`.

Si può anche trovare la cartella architecture creata in `lvgl_yocto_guide/ build/tmp/work/cortexa53-poky-linux/lvgl` ma queste cartelle sono vuote poiché è stato eseguito solo il fetch.

Unpack (do_unpack)

```
bitbake lvgl -c unpack
```

Il task "unpack" scompatta il pacchetto scaricato co task Fetch.

Nella cartella `lvgl_yocto_guide/build/tmp/work/cortexa53-poky-linux/lvgl/9.1.0`, ora si può trovare il codice sorgente in `git` (come definito nella ricetta). Si vedranno anche le patch che verranno applicate nel passaggio successivo. Quindi, per ora, i sorgenti non sono modificati e sono uguali al commit specificato.

Patch (do_patch)

```
bitbake lvgl -c patch
```

Il task "patch" individua i file delle patch e applica le patch ai sorgenti, se disponibili. Questo è un task facoltativo, eseguito se la patch è disponibile.

Il file della patch è definito anche nella variabile **SRC_URI**. Per default viene eseguito nella directory sorgente corrente **${S}**.

Configure (do_configure)

```
bitbake lvgl -c configure
```

Il task di configurazione configura il codice sorgente abilitando e disabilitando eventuali opzioni di build e di configurazione per il software in fase di compilazione prima della compilazione, se disponibile.

Questo è un passaggio facoltativo, eseguito se disponibile.

In questo caso, crea una directory di build e richiama CMake per configurare il progetto, specificando le opzioni di build e i path in base alle variabili nella ricetta. Genera i Makefile o i file di progetto necessari per la build. Inoltre, ci sono operazioni aggiunte al task in `lv-conf.inc`.

Quindi, alla fine del task, in `lvgl_yocto_guide/build/tmp/work/ cortexa53-poky-linux/lvgl/9.1.0`, troverete una cartella `build` che è stata generata eseguendo il comando CMake, ma non è ancora stato compilato nulla. Inoltre, le sysroot contengono tutto il necessario per compilare la libreria lvgl.

Compile (do_compile)

```
bitbake lvgl -c compile
```

Il task Compilation compila il codice sorgente se sono disponibili passaggi di compilazione e genera un file binario.

Questo è un passaggio facoltativo, eseguito se la compilazione è disponibile.

Se sono presenti passaggi di compilazione, questi vengono definiti nella funzione do_compile() di bitbake.

Come nel task precedente, questa operazione viene gestita da `inherit cmake`.

Nella cartella di build, ora è possibile visualizzare la libreria compilata. I file `.so` sono disponibili in `lvgl_yocto_guide/build/tmp/work/ cortexa53-poky-linux/lvgl/9.1.0/build/lib`.

Una volta completata questa attività, tutto è pronto per l'installazione.

Install (do_install)

```
bitbake lvgl -c install
```

Il task Install copia i file da impacchettare nell'area di archiviazione **${D}**. Questo viene eseguito con la directory di lavoro corrente **${S}**, che è la directory di compilazione.

Crea la directory necessaria per i file header nella directory di installazione di destinazione. Installa il file header `lv_conf.h` dalla directory sorgente in quella di inclusione appropriata nel path di destinazione, assicurandosi che abbia i permessi corretti.

Anche la libreria lvgl (file `.so`) è pronta per essere installata nell'immagine finale. È stata creata una nuova cartella `image`.

Importante

Nel file `build/conf/local.conf`, aggiungere queste 2 righe alla fine del file:

```
IMAGE_INSTALL:append = " lvgl"
TOOLCHAIN_HOST_TASK:append = " lvgl"
```

Questo aggiungerà la libreria lvgl all'immagine generata e la aggiungerà anche all'SDK host che genereremo in seguito.

Con queste modifiche, ora si può eseguire nuovamente la ricetta dell'immagine:

```
bitbake core-image-base
```

Questo eseguirà tutte le attività descritte in precedenza.

Se tutto è andato bene, si troverà il file `build/tmp/deploy/ rpm/cortexa53/lvgl-9.1.0-r0.cortexa53.rpm` e altri file rpm relativi a lvgl.

Generazione dell'SDK

La generazione di un Software Development Kit (SDK) in Yocto svolge diverse funzioni importanti, in particolare nello sviluppo embedded:

- **Ambiente di sviluppo**: L'SDK fornisce agli sviluppatori un ambiente di sviluppo pronto all'uso, personalizzato per una specifica piattaforma hardware di destinazione. Questo include gli strumenti, le librerie e gli header necessari per creare applicazioni eseguibili sul dispositivo target.
- **Cross-Compilazione**: L'SDK consente agli sviluppatori di compilare in modo incrociato le applicazioni da una macchina host (tipicamente con architettura x86 o x86_64) per un'architettura diversa (ad esempio, ARM). Questo è essenziale per i sistemi embedded, dove lo sviluppo avviene spesso su macchine più potenti.
- **Sviluppo Semplificato**: Impacchettando tutti i componenti necessari, l'SDK semplifica il processo di configurazione di un ambiente di sviluppo. Gli sviluppatori non devono installare e configurare manualmente strumenti e librerie; tutto il necessario è incluso nell'SDK.
- **Ambiente di Build Coerente**: L'SDK garantisce che gli sviluppatori lavorino con le stesse versioni e gli stessi strumenti utilizzati nella build di Yocto, il che aiuta a evitare problemi di compatibilità e garantisce che le applicazioni si comportino come previsto sul dispositivo target.

Per generare un SDK dell'ambiente, eseguire il seguente comando:

```
bitbake core-image-base -c populate_sdk
```

Questo creerà uno script `build/tmp/deploy/sdk/ poky-glibc-x86_64-core-image-base-cortexa53-raspberrypi3-64-toolchain-5.0.4. sh`. Questo script consente di installare l'SDK dove si preferisce. Ecco l'output di esecuzione dello script

```
$ ./sdk/poky-glibc-x86_64-core-image-base-cortexa53-raspberrypi3-64-
↪toolchain-5.0.4.sh
Poky (Yocto Project Reference Distro) SDK installer version 5.0.4
==================================================================
Enter target directory for SDK (default: /opt/poky/5.0.4): /opt/poky/sdk-
↪with-lvgl
You are about to install the SDK to "/opt/poky/sdk-with-lvgl". Proceed [Y/
↪n]? y
```

Per assicurarsi che l'SDK sia stato generato con lvgl installato, si va al path in cui è stato estratto l'SDK e si trovano tutti i file lvgl:

```
cd /opt/poky/5.0.4/sysroots/cortexa53-poky-linux
find . -name "*lvgl*"
```

I file `.so` che si troveranno dipenderanno dalla configurazione LVGL utilizzata.

Ora, per utilizzare l'ambiente SDK e cross-compilare un'applicazione:

```
source /opt/poke/5.0.4/environment-setup-cortexa53-poky-linux
```

> **Nota**
>
> I tool disponibili nella libreria LVGL dipenderanno dalla configurazione di build con cui è stata realizzata la ricetta.

Ricetta personalizzata

Fino a questa sezione, tutto era già stato preparato. Abbiamo utilizzato ricette esistenti. L'obiettivo qui è creare una ricetta da zero e aggiungere il binario generato all'immagine.

Creare un layer

Per prima cosa, si crea un layer e lo si aggiunge al file di configurazione.

```
bitbake-layers create-layer ../sources/meta-mylvgl
bitbake-layers add-layer ../sources/meta-mylvgl
```

Nella cartella `sources`, è stata creata una nuova cartella: `meta-mylvgl` e l'albero delle directory dovrebbe apparire come segue.

```
├── conf
│   └── layer.conf
├── COPYING.MIT
├── README
└── recipes-example
    └── example
        └── example_0.1.bb
```

Creare una ricetta

Seguendo questa struttura, si crea una cartella contenente le ricette per compilare una o più applicazioni utilizzando lvgl.

```
cd ../sources/meta-mylvgl
mkdir -p recipes-lvglapp/lvgl-fbdev-benchmark/files
touch recipes-lvglapp/lvgl-fbdev-benchmark/lvglbenchmarkfbdev_2.4.bb
```

Ci concentreremo su un'applicazione che clonerà un repository git lvgl e lo patcherà per le nostre esigenze.

Il contenuto di `recipes-lvglapp/lvgl-fbdev-benchmark/lvglbenchmarkfbdev_2.4. bb`

```
DESCRIPTION = "My C++ lvgl app with CMake"

LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;
↪md5=0835ade698e0bcf8506ecda2f7b4f302"

# Update SRC_URI to point to the GitHub repository
SRC_URI = "gitsm://github.com/lvgl/lv_port_linux.git;branch=master;
↪protocol=https \
           file://0001-change-config-file-to-run-fbdev-with-2-threads.
↪patch \
           file://0002-adapt-CMakeLists-file-to-compile-and-link-fbdev.
↪patch \
          "

S = "${WORKDIR}/git"

inherit cmake

CMAKE_PROJECT_NAME = "lvgl_app"
CMAKE_PROJECT_VERSION = "2.0"

do_install() {
    install -d ${D}${bindir}
    install -m 0755 ${S}/bin/lvglbenchmark ${D}${bindir}
}
```

I sorgenti provengono dal repository `lv_port_linux`. Applichiamo 2 patch per modificare `CMakeLists.txt` e `lv_conf.h`.

Patch 1

Creazione del primo file patch

```
touch 0001-change-config-file-to-run-fbdev-with-2-threads.patch
```

Contenuto di `0001-change-config-file-to-run-fbdev-with-2-threads.patch`

```
From d49d475d57f69d6172a6b38095ebf3b887f0e731 Mon Sep 17 00:00:00 2001
From: Gabriel Catel Torres <gabriel.catel@edgemtech.ch>
Date: Tue, 1 Oct 2024 10:28:55 +0200
Subject: [PATCH] change config file to run fbdev with 2 threads

---
lv_conf.h | 34 +++++++++++++++++++----------------
1 file changed, 17 insertions(+), 17 deletions(-)

diff --git a/lv_conf.h b/lv_conf.h
index 62a834f..58fbe7a 100644
--- a/lv_conf.h
+++ b/lv_conf.h
@@ -39,9 +39,9 @@
  * - LV_STDLIB_RTTHREAD:    RT-Thread implementation
  * - LV_STDLIB_CUSTOM:      Implement the functions externally
  */
-#define LV_USE_STDLIB_MALLOC    LV_STDLIB_BUILTIN
-#define LV_USE_STDLIB_STRING    LV_STDLIB_BUILTIN
-#define LV_USE_STDLIB_SPRINTF   LV_STDLIB_BUILTIN
+#define LV_USE_STDLIB_MALLOC    LV_STDLIB_CLIB
+#define LV_USE_STDLIB_STRING    LV_STDLIB_CLIB
+#define LV_USE_STDLIB_SPRINTF   LV_STDLIB_CLIB

#define LV_STDINT_INCLUDE       <stdint.h>
#define LV_STDDEF_INCLUDE       <stddef.h>
@@ -89,7 +89,7 @@
  * - LV_OS_WINDOWS
  * - LV_OS_MQX
  * - LV_OS_CUSTOM */
-#define LV_USE_OS   LV_OS_NONE
+#define LV_USE_OS   LV_OS_PTHREAD

#if LV_USE_OS == LV_OS_CUSTOM
    #define LV_OS_CUSTOM_INCLUDE <stdint.h>
@@ -117,12 +117,12 @@
  * and can't be drawn in chunks. */

/* The target buffer size for simple layer chunks. */
-#define LV_DRAW_LAYER_SIMPLE_BUF_SIZE    (24 * 1024)   /* [bytes] */
+#define LV_DRAW_LAYER_SIMPLE_BUF_SIZE    (512 * 1024)   /* [bytes] */

/* The stack size of the drawing thread.
  * NOTE: If FreeType or ThorVG is enabled, it is recommended to set it to␣
↪32KB or more.
  */
-#define LV_DRAW_THREAD_STACK_SIZE    (8 * 1024)   /* [bytes] */
```

(continues on next page)

(continua dalla pagina precedente)

```
+#define LV_DRAW_THREAD_STACK_SIZE    (32 * 1024)   /* [bytes] */

#define LV_USE_DRAW_SW 1
#if LV_USE_DRAW_SW == 1
@@ -147,7 +147,7 @@
  /* Set the number of draw unit.
      * > 1 requires an operating system enabled in `LV_USE_OS`
      * > 1 means multiple threads will render the screen in parallel */
-    #define LV_DRAW_SW_DRAW_UNIT_CNT    1
+    #define LV_DRAW_SW_DRAW_UNIT_CNT    2

    /* Use Arm-2D to accelerate the sw render */
    #define LV_USE_DRAW_ARM2D_SYNC      0
@@ -979,8 +979,8 @@
#define LV_USE_LINUX_FBDEV      1
#if LV_USE_LINUX_FBDEV
    #define LV_LINUX_FBDEV_BSD           0
-    #define LV_LINUX_FBDEV_RENDER_MODE   LV_DISPLAY_RENDER_MODE_PARTIAL
-    #define LV_LINUX_FBDEV_BUFFER_COUNT  0
+    #define LV_LINUX_FBDEV_RENDER_MODE   LV_DISPLAY_RENDER_MODE_DIRECT
+    #define LV_LINUX_FBDEV_BUFFER_COUNT  2
    #define LV_LINUX_FBDEV_BUFFER_SIZE   60
#endif

@@ -1069,19 +1069,19 @@
#define LV_USE_DEMO_WIDGETS 1

/* Demonstrate the usage of encoder and keyboard */
-#define LV_USE_DEMO_KEYPAD_AND_ENCODER 1
+#define LV_USE_DEMO_KEYPAD_AND_ENCODER 0

/* Benchmark your system */
#define LV_USE_DEMO_BENCHMARK 1

/* Render test for each primitives. Requires at least 480x272 display */
-#define LV_USE_DEMO_RENDER 1
+#define LV_USE_DEMO_RENDER 0

/* Stress test for LVGL */
-#define LV_USE_DEMO_STRESS 1
+#define LV_USE_DEMO_STRESS 0

/* Music player demo */
-#define LV_USE_DEMO_MUSIC 1
+#define LV_USE_DEMO_MUSIC 0
#if LV_USE_DEMO_MUSIC
    #define LV_DEMO_MUSIC_SQUARE    0
    #define LV_DEMO_MUSIC_LANDSCAPE 0
@@ -1091,16 +1091,16 @@
#endif

/* Flex layout demo */
-#define LV_USE_DEMO_FLEX_LAYOUT     1
+#define LV_USE_DEMO_FLEX_LAYOUT     0

/* Smart-phone like multi-language demo */
```

(continues on next page)

(continua dalla pagina precedente)

```
-#define LV_USE_DEMO_MULTILANG       1
+#define LV_USE_DEMO_MULTILANG       0

/* Widget transformation demo */
-#define LV_USE_DEMO_TRANSFORM       1
+#define LV_USE_DEMO_TRANSFORM       0

/* Demonstrate scroll settings */
-#define LV_USE_DEMO_SCROLL          1
+#define LV_USE_DEMO_SCROLL          0

/* Vector graphic demo */
#define LV_USE_DEMO_VECTOR_GRAPHIC  0
--
2.34.1
```

Patch 2

Creazione del primo file patch

```
touch 0002-adapt-CMakeLists-file-to-compile-and-link-fbdev.patch
```

Contenuto di `0002-adapt-CMakeLists-file-to-compile-and-link-fbdev.patch`

```
From ad464e0683aced74085fb89895b8d040ecad0206 Mon Sep 17 00:00:00 2001
From: Gabriel Catel Torres <gabriel.catel@edgemtech.ch>
Date: Tue, 1 Oct 2024 10:31:29 +0200
Subject: [PATCH] adapt CMakeLists file to compile and link only for fbdev

---
CMakeLists.txt | 17 +++++------------
1 file changed, 5 insertions(+), 12 deletions(-)

diff --git a/CMakeLists.txt b/CMakeLists.txt
index 658193f..ad56cc2 100644
--- a/CMakeLists.txt
+++ b/CMakeLists.txt
@@ -1,8 +1,8 @@
cmake_minimum_required(VERSION 3.10)
project(lvgl)

-set(CMAKE_C_STANDARD 99)#C99 # lvgl officially support C99 and above
-set(CMAKE_CXX_STANDARD 17)#C17
+set(CMAKE_C_STANDARD 99)
+set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

set(EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
@@ -10,15 +10,8 @@ set(EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)
add_subdirectory(lvgl)
target_include_directories(lvgl PUBLIC ${PROJECT_SOURCE_DIR})

-add_executable(main main.c mouse_cursor_icon.c)
+add_executable(lvglbenchmark main.c mouse_cursor_icon.c)

-include(${CMAKE_CURRENT_LIST_DIR}/lvgl/tests/FindLibDRM.cmake)
```

(continues on next page)

(continua dalla pagina precedente)

```
-include_directories(${Libdrm_INCLUDE_DIRS})
-
-find_package(SDL2)
-find_package(SDL2_image)
-include_directories(${SDL2_INCLUDE_DIRS} ${SDL2_IMAGE_INCLUDE_DIRS})
-
-target_link_libraries(main lvgl lvgl::examples lvgl::demos lvgl::thorvg $
↪{SDL2_LIBRARIES} ${SDL2_IMAGE_LIBRARIES} ${Libdrm_LIBRARIES} m pthread)
-add_custom_target (run COMMAND ${EXECUTABLE_OUTPUT_PATH}/main DEPENDS␣
↪main)
+target_link_libraries(lvglbenchmark lvgl lvgl::examples lvgl::demos␣
↪lvgl::thorvg m pthread)
+add_custom_target (run COMMAND ${EXECUTABLE_OUTPUT_PATH}/lvglbenchmark␣
↪DEPENDS main)

--
2.34.1
```

Compilare la ricetta

Ora si dovrebbe essere in grado di visualizzare la ricetta con l'elenco delle ricette esistenti.

```
bitbake-layers show-recipes | grep lvglbenchmarkfbdev
```

Si aggiunge la ricetta al progetto. Modificare il file `build/conf/local.conf` aggiungendo questa riga.

```
IMAGE_INSTALL:append = " lvglbenchmarkfbdev"
```

Ora compilare l'immagine.

```
bitbake core-image-base
```

Eseguire questo comando per assicurarsi che il binario sia stato generato e installato nel rootfs.

```
# Run this command in build/tmp folder
find . -wholename "*bin/lvglbench*"
```

Terminologia dello Yocto Project

Iniziare a usare Yocto può essere complicato. Sono molti i termini utilizzati, specifici per Yocto e per l'ambiente Bitbake.

Un elenco di termini e definizioni che gli utenti alle prime armi con l'ambiente di sviluppo del progetto Yocto potrebbero trovare utili è disponibile qui.

Glossario delle Variabili di Yocto

Questo capitolo elenca le variabili di base utilizzate nella guida LVGL di Yocto e fornisce una panoramica della loro funzione e del loro contenuto.

Un glossario più completo delle variabili è disponibile nello Yocto Variable Glossary. Questa sezione tratta molte variabili utilizzate nel sistema di build OpenEmbedded.

S

Posizione nella "Build Directory" in cui risiede il codice sorgente della ricetta spacchettata. Per default, questa directory è ${WORKDIR}/${BPN}-${PV}, dove ${BPN} è il nome della ricetta di base e ${PV} è la versione della ricetta. Se il tarball sorgente estrae il codice in una directory con un nome diverso da ${BPN}-${PV}, o se il codice sorgente viene recuperato da un SCM come Git o Subversion, è necessario impostare S nella ricetta in modo che il sistema di build di OpenEmbedded sappia dove trovare il codice sorgente decompresso.

Ad esempio, si supponga una "Source Directory" di primo livello denominata poky e una "Build Directory" di default in poky/build. In questo caso, la directory di lavoro utilizzata dal sistema di build per conservare la ricetta decompressa per db è la seguente:

```
poky/build/tmp/work/qemux86-poky-linux/db/5.1.19-r3/db-5.1.19
```

Il codice sorgente decompresso si trova nella cartella db-5.1.19.

Il prossimo esempio presuppone un repository Git. Per default, i repository Git vengono clonati in ${WORKDIR}/git during do_fetch. Poiché questo path è diverso dal valore predefinito di S, è necessario impostarlo specificamente in modo che il sorgente possa essere individuato:

```
SRC_URI = "git://path/to/repo.git;branch=main"
S = "${WORKDIR}/git"
```

D

Directory di destinazione. La posizione nella Build Directory in cui i componenti vengono installati dal task do_install. Questa posizione di defaults è:

```
${WORKDIR}/image
```

> **Nota**
>
> I task che leggono o scrivono in questa directory devono essere eseguiti con fakeroot.

B

La directory all'interno della Build Directory in cui il sistema di build OpenEmbedded inserisce gli oggetti generati durante il processo di build di una ricetta. Per default, questa directory è la stessa della directory S, definita come:

```
S = "${WORKDIR}/${BP}"
```

È possibile separare la directory (S) e la directory puntata dalla variabile B. La maggior parte delle ricette basate su Autotools supporta la separazione di queste directory. Il sistema di build utilizza per default directory separate per gcc e alcune ricette del kernel.

WORKDIR

Il percorso della directory di lavoro in cui il sistema di build di OpenEmbedded crea una ricetta. Questa directory si trova all'interno della struttura di directory TMPDIR ed è specifica della ricetta in fase di compilazione e del sistema per cui viene compilata.

La directory WORKDIR è definita come segue:

```
${TMPDIR}/work/${MULTIMACH_TARGET_SYS}/${PN}/${EXTENDPE}${PV}-${PR}
```

La directory effettiva dipende da diversi fattori:

- **TMPDIR**: Directory di output della build di livello superiore
- **MULTIMACH_TARGET_SYS**: Identificativo del sistema target

- **PN**: Nome della ricetta
- **EXTENDPE**: Epoch: se PE non è specificato, come solitamente accade per la maggior parte delle ricette, EXTENDPE è vuoto.
- **PV**: Versione della ricetta
- **PR**: Revisione della ricetta

Ad esempio, supponiamo una Source Directory di livello superiore denominata poky, una Build Directory predefinita in poky/build e un sistema target qemux86-poky-linux. Inoltre, supponiamo che la ricetta si chiami foo_1.3.0-r0.bb. In questo caso, la directory di lavoro utilizzata dal sistema di build per compilare il pacchetto sarebbe la seguente:

```
poky/build/tmp/work/qemux86-poky-linux/foo/1.3.0-r0
```

PN

Questa variabile può avere due funzioni distinte a seconda del contesto: un nome di ricetta o un nome di pacchetto risultante.

PN si riferisce a un nome di ricetta nel contesto di un file utilizzato dal sistema di build di OpenEmbedded come input per creare un pacchetto. Il nome viene normalmente estratto dal nome del file della ricetta. Ad esempio, se la ricetta si chiama expat_2.0.1.bb, il valore di default di PN sarà “expat”.

La variabile si riferisce a un nome di pacchetto nel contesto di un file creato o prodotto dal sistema di build di OpenEmbedded.

Se applicabile, la variabile PN contiene anche eventuali suffissi o prefissi speciali. Ad esempio, utilizzando bash per compilare pacchetti per la macchina nativa, PN è bash-native. Utilizzando bash per compilare pacchetti per il target e per Multilib, PN sarebbe rispettivamente bash e lib64-bash.

PR

La revisione della ricetta. Il valore di default per questa variabile è "r0". Le revisioni successive della ricetta hanno convenzionalmente i valori "r1", "r2" e così via. Quando il PV aumenta, il PR viene convenzionalmente reimpostato a "r0".

> **Nota**
>
> Il sistema di build di OpenEmbedded non ha bisogno dell'ausilio del PR per sapere quando ri-buildare una ricetta. Il sistema di build utilizza i checksum di input dei task insieme ai meccanismi di "stamp" [timbro] e cache di stato condivisa.

La variabile PR diventa significativa principalmente quando un gestore di pacchetti installa dinamicamente i pacchetti su un'immagine già compilata. In questo caso, il PR, che è il valore di default di PKGR, aiuta il gestore di pacchetti a distinguere quale pacchetto sia il più recente nei casi in cui molti pacchetti abbiano lo stesso PV (ad esempio, PKGV). Un componente con molti pacchetti con lo stesso PV di solito significa che tutti i pacchetti installano la stessa versione upstream, ma con pacchetti di versioni successive (PR) che includono correzioni al packaging.

> **Nota**
>
> Il PR non deve essere aumentato per le modifiche che non alterano il contenuto o i metadati del pacchetto.

Poiché la gestione manuale delle PR può essere macchinosa e soggetta a errori, esiste una soluzione automatizzata. Per ulteriori informazioni, consultare la sezione "working with a pr service" nello Yocto Project Development Tasks Manual.

PV

La versione della ricetta. La versione viene normalmente estratta dal nome del file della ricetta. Ad esempio, se la ricetta si chiama expat_2.0.1.bb, il valore predefinito di PV sarà "2.0.1". PV generalmente non viene sovrascritto all'interno di una ricetta, a meno che non si stia compilando una versione instabile (ad esempio, di sviluppo) da un repository di codice sorgente (ad esempio Git o Subversion).

PV è il valore predefinito della variabile PKGV.

4.9.10 Zephyr

Cos'è Zephyr?

Zephyr è un sistema operativo real-time (RTOS) open source facile da implementare, proteggere, connettere e gestire. Dispone di un set crescente di librerie software utilizzabili in diverse applicazioni e settori industriali, come l'IoT industriale, i dispositivi indossabili, il machine learning e altro ancora. Zephyr è progettato con particolare attenzione all'ampio supporto di chipset, alla sicurezza, all'affidabilità, alle release con supporto a lungo termine e a un ecosistema open source in crescita.

Punti di Forza di Zephyr

- **Piccolo** - Funziona su microcontrollori con soli 8 kB di Flash e 5 kB di RAM.
- **Scalabile** - Utilizzabile per sistemi multicore complessi.
- **Personalizzabile** - Supporto immediato per oltre 500 schede e elevata portabilità.
- **Sicuro** - Progettato pensando alla sicurezza e alla protezione, offre supporto "Long-term" [a lungo termine].
- **Ecosistema** - Zephyr non fornisce solo il kernel RTOS, ma anche strumenti di sviluppo, device driver, connettività, logging, tracing, gestione dell'alimentazione e molto altro.
- **Disaccoppiamento** - Sfrutta il devicetree per descrivere e configurare il sistema target.
- **Conforme** - Le app sono eseguibili come applicazioni Linux native, semplificando il debug e la profilazione.

Come eseguire LVGL su Zephyr?

Per configurare l'ambiente di sviluppo, consultare la guida introduttiva.

Dopo aver completato la configurazione sopra descritta, si possono consultare tutti gli esempi forniti per diverse schede. Si può consultare l'elenco delle schede disponibili usando:

```
$ west boards
```

Dopo aver scelto una scheda, si può compilare una delle demo LVGL per essa. Qui utilizziamo la scheda `native_posix`, che consente di eseguire l'applicazione sul sistema host compatibile con Posix:

```
$ west build -b native_posix samples/modules/lvgl/demos
```

Per eseguire l'applicazione sul proprio host:

```
$ west build -t run
```

Se è stata scelta una qualsiasi delle altre schede supportate, si può flashare sul dispositivo con:

```
$ west flash
```

Per compilare una qualsiasi delle altre applicazioni demo, consultare gli esempi README.

Sfruttare le funzionalità di Zephyr

Shell

Zephyr include una potente implementazione della shell abilitabile con i simboli Kconfig `CONFIG_SHELL` e `CONFIG_LV_Z_SHELL` (le demo precedenti la abilitano di default).

La shell offre la possibilità di abilitare/disabilitare i dispositivi LVGL:

```
# Create a new monkey with the given indev type
uart$ lvgl monkey create [pointer|keypad|button|encoder]

# Enable/Disable a monkey
uart$ lvgl monkey set <index> <inactive/active>
```

Questo è utile per verificare la presenza di "memory leak" [perdite di memoria] e altri bug nell'applicazione. A proposito di memory leak, è anche possibile acquisire statistiche sulla memoria utilizzata da LVGL.

```
uart$ lvgl stats memory
```

Per maggiori dettagli, consultare la documentazione della shell.

Devicetree

Zephyr utilizza il linguaggio di descrizione devicetree per creare e gestire dispositivi di input LVGL.

Le descrizioni dei binding degli pseudo dispositivi sono disponibili all'indirizzo:

- button input
- pointer input
- encoder input
- keypad input

In sostanza, questi bufferizzano l'evento `input_event` generato dal dispositivo puntato dal phandle `input` oppure, se lasciato vuoto, il binding cattura tutti gli eventi indipendentemente dalla sorgente. Non è necessario istanziare o gestire i dispositivi manualmente, poiché vengono creati all'avvio dell'applicazione prima dell'esecuzione di `main()`.

La maggior parte delle schede o degli shield dotati di display o connettore per display hanno il dispositivo di input del puntatore già dichiarato:

```
lvgl_pointer {
    compatible = "zephyr,lvgl-pointer-input";
    input = <&ft5336_touch>;
};
```

È possibile accedere al `lv_indev_t` di lvgl sottostante per la configurazione. Esempio con il dispositivo encoder per assegnare un `lv_group_t`:

```
const struct device *lvgl_encoder = DEVICE_DT_GET(DT_COMPAT_GET_ANY_STATUS_
↪OKAY(zephyr_lvgl_encoder_input));

lv_obj_t *arc;
lv_group_t *arc_group;

arc = lv_arc_create(lv_screen_active());
lv_obj_align(arc, LV_ALIGN_CENTER, 0, 0);
lv_obj_set_size(arc, 150, 150);

arc_group = lv_group_create();
```

(continues on next page)

(continua dalla pagina precedente)

```
lv_group_add_obj(arc_group, arc);
lv_indev_set_group(lvgl_input_get_indev(lvgl_encoder), arc_group);
```

Kconfig

Oltre ad abilitare la shell, è possibile utilizzare Kconfig per ottimizzare il footprint dell'applicazione.

```
# Size of the memory region from which lvgl memory is allocated
CONFIG_LV_Z_MEM_POOL_SIZE=8192

# Do not include every widget/theme by default, enable them as needed.
CONFIG_LV_CONF_MINIMAL=y
```

Gli overlay possono essere utilizzati per abilitare/disabilitare funzionalità per schede specifiche o target di build. Per ulteriori informazioni, consultare la guida allo sviluppo di applicazioni.

Ottimizzazione delle prestazioni in LVGL

Per ottimizzare le prestazioni di LVGL, è possibile configurare diverse opzioni `kconfig`:

- **CONFIG_LV_Z_VDB_SIZE**: Imposta la dimensione del buffer di rendering come percentuale dell'area di visualizzazione, regolabile dall'1% al 100%. Buffer più grandi possono migliorare le prestazioni, soprattutto se utilizzati con **CONFIG_LV_Z_FULL_REFRESH**.
- **CONFIG_LV_Z_DOUBLE_VDB**: Abilita l'utilizzo di due buffer di rendering, consentendo il rendering parallelo e il flushing dei dati, migliorando così la reattività e riducendo la latenza.
- **CONFIG_LV_Z_VDB_ALIGN**: Garantisce che il buffer di rendering sia correttamente allineato, il che è fondamentale per un accesso efficiente alla memoria in base alla profondità del colore.
- **CONFIG_LV_Z_VBD_CUSTOM_SECTION**: Consente di posizionare i buffer di rendering in una sezione di memoria personalizzata (ad esempio, `.lvgl_buf`), utile per sfruttare tipi di memoria specifici come quella strettamente accoppiata o esterna per migliorare le prestazioni.

Opzioni specifiche per Zephyr ≤ 3.7.0

Per le versioni di Zephyr 3.7.0 e precedenti, sono disponibili opzioni aggiuntive per gestire il flushing dei frame LVGL:

- **CONFIG_LV_Z_FLUSH_THREAD**: Abilita il flushing dei frame LVGL in un thread separato, consentendo al thread principale di continuare a eseguire il rendering del frame successivo simultaneamente. Questa opzione può essere disabilitata se non si desidera ottenere un miglioramento delle prestazioni.
  - **CONFIG_LV_Z_FLUSH_THREAD_STACK_SIZE**: Specifica la dimensione dello stack per il thread di flushing, con un valore di default di 1024 byte.
  - **CONFIG_LV_Z_FLUSH_THREAD_PRIO**: Imposta la priorità del thread di flushing, con un valore predefinito di 0, che indica la priorità cooperativa.

Per le versioni più recenti di Zephyr, è possibile utilizzare OSAL (Operating System Abstraction Layer), che si occupa del flushing.

Dove si possono trovare maggiori informazioni?

- Documentazione di Zephyr
- Mailing list di Zephyr
- Server Discord di Zephyr

4.10 Board

4.10.1 Toradex

Informazioni su Toradex

Toradex è specializzata nella produzione di soluzioni di elaborazione embedded di alta qualità basate su unità System-on-Module (SoM). Queste schede e moduli sono progettati per soddisfare le esigenze di applicazioni industriali e commerciali, offrendo affidabilità, longevità e scalabilità. Sfruttando framework standard di settore come lo Yocto Project, Toradex consente ai clienti di creare sistemi embedded personalizzati con facilità, garantendo compatibilità e flessibilità.

Portfolio Prodotti

Toradex offre un'ampia gamma di System-on-Module (SoM) e schede carrier progettate per diverse applicazioni e settori. Ecco una panoramica di base per aiutare a navigare nel portfolio.

Aquila

Aquila è un nuovo standard System on Module che espande il portfolio Toradex a nuovi livelli, offrendo prestazioni senza pari per carichi di lavoro CPU, machine learning e visione artificiale ad alta intensità di calcolo.

Per soddisfare i requisiti di intelligenza artificiale di nuova generazione in ambienti industriali e in altri ambienti difficili, Aquila introduce un connettore collaudato: il connettore board-to-board a 400 pin, che rende la scheda estremamente affidabile, soprattutto in questi ambienti.

Computer on Module Aquila:

- Aquila iMX95
- Aquila AM69

Schede Carrier Aquila:

- Clover
- Aquila Development Board

Apalis

Apalis è una famiglia scalabile di System on Module (SoM) / Computer on Module (CoM) che mira a fornire elevate prestazioni in un fattore di forma compatto. Apalis consente lo sviluppo di prodotti avanzati e robusti in tempi rapidi e con rischi ridotti. I moduli Apalis raggiungono le prestazioni più elevate nella gamma di prodotti Toradex e offrono il maggior numero di interfacce di qualsiasi altra linea di prodotti sui connettori MXM a 314 pin. I moduli Apalis sono adatti a un ampio numero di applicazioni in settori quali sanità, automazione industriale, robotica, smart city e molti altri. I moduli Apalis sono supportati dall'offerta software Toradex e dal ricco ecosistema di altri prodotti e servizi dell'azienda.

Computer on Module Apalis:

- Apalis iMX8
- Apalis iMX6
- Apalis T30

Schede Carrier Apalis:

- Ixora
- Apalis Evaluation Board

Colibri

La famiglia The Colibri Arm® è composta da System on Module (SoM) / Computer on Module (CoM) di dimensioni SODIMM. La famiglia offre moduli miniaturizzati basati su NXP® i.MX 8X, i.MX 6ULL, i.MX 7, i.MX 6, Vybrid e NVIDIA® Tegra SoCs. Tutti questi moduli sono compatibili a livello di pin, garantendo la scalabilità delle applicazioni sviluppate su di essi. La famiglia Colibri offre progetti ottimizzati in termini di costi e prestazioni. È inoltre disponibile un'ampia gamma di schede carrier e accessori compatibili con Colibri, come fotocamere e display, per un facile sviluppo di prodotti sulla famiglia di moduli Colibri Arm.

Computer on Module Colibri:

- Colibri iMX8X
- Colibri T30
- Colibri T20
- Colibri iMX6
- Colibri iMX7
- Colibri iMXiMX6ULL8X
- Colibri VF61
- Colibri VF50

Schede Carrier Colibri:

- Colibri Evaluation Board
- Iris Carrier Board
- Viola Carrier Board
- Aster Carrier Board

Verdin

Verdin è uno standard System on Module che amplia i fattori di forma di successo Colibri e Apalis. Offre un set di interfacce rivisitato, moderno e funzionale, puntando su facilità d'uso, robustezza e compatibilità con le esigenze future.

Verdin ha dimensioni simili a Colibri, ma con un numero di pin maggiore. Un connettore edge SODIMM DDR4 offre una connessione conveniente, altamente affidabile e resistente a urti e vibrazioni.

Computer on Module Verdin:

- Verdin iMX8M Plus
- Verdin iMX8M Mini
- Verdin AM62
- i.MX 95 Verdin Evaluation Kit
- Verdin iMX95

Schede Carrier Verdin:

- Verdin Development Board con HDMI Adapter
- Dahlia Carrier Board con HDMI Adapter
- Yavia
- Mallow Carrier Board
- Ivy Carrier Board

TorizonOS

TorizonOS è un sistema operativo basato su Linux sviluppato da Toradex, pensato appositamente per i sistemi embedded. È progettato per semplificare lo sviluppo, l'implementazione e la manutenzione delle applicazioni, in particolare in ambito industriale e IoT. Combinando la facilità d'uso con potenti funzionalità come la containerizzazione e gli aggiornamenti over-the-air (OTA), TorizonOS aiuta gli sviluppatori ad accelerare il time-to-market, garantendo al contempo prestazioni e sicurezza elevate per le applicazioni embedded.

Una guida di TorizonOS per sviluppare un'applicazione utilizzando LVGL è disponibile nella sezione *Torizon OS*.

Ulteriori informazioni sono disponibili nella documentazione di Torizon.

Esempi di Toradex

Sono disponibili repository pronti all'uso. Cliccare qui per consultarli.

4.10.2 Riverdi

Riverdi è specializzata nella produzione di display di alta qualità e delle relative schede. Offrono soluzioni che spaziano dai semplici pannelli display ai display intelligenti, con tutto ciò che sta nel mezzo.

- Display embedded STM32
- Display per Computer Single Board
- Display intelligenti EVE
- Display HDMI
- Evaluation Board
- Display RGB, LVDS, MIPI DSI LCD

Display embedded STM32

I display embedded STM32 sono dotati di un MCU STM32 integrato che consente di integrare un'intera applicazione nel dispositivo. LVGL è particolarmente adatto a queste schede.

lv_port_riverdi_stm32u5

lv_port_riverdi_stm32u5 is a port repo of LVGL which supports all 5-inch Riverdi STM32 Embedded Displays.

- RVT50HQSNWC00-B
- RVT50HQSNWC00
- RVT50HQSFWCA0
- RVT50HQSNWCA0
- RVT50HQSFWN00
- RVT50HQSNWN00

Supporta nativamente la GPU NeoChrom integrata grazie al supporto LVGL per *Nema GFX*. NemaVG --- un'estensione del supporto base di Nema, è implementato da questa MCU.

L'MCU è dotato di una periferica driver LCD che LVGL utilizza per aggiornare il display. Consultare la documentazione del driver *LTDC* per maggiori informazioni e per personalizzarne l'utilizzo.

Display per Single-Board Computer

I Single-Board Computer Displays sono pronti per l'uso con le schede carrier Dahlia e Mallow *Toradex*. Infatti, queste schede carrier sono compatibili con tutti i display Riverdi MIPI-DSI a 34 pin.

Altri prodotti

Gli **EVE Intelligent Display** sono dotati di un controller grafico IC Bridgetek EVE, in modo che il display possa essere controllato utilizzando comandi di disegno di alto livello su un'interfaccia a larghezza di banda inferiore rispetto a RGB, MIPI, ecc.

Gli **HDMI Displays** sono dotati di un'interfaccia HDMI per un'integrazione semplificata con PC o SBC. sport an HDMI interface for streamlined integration with PCs or SBCs. Questi display hanno comunque un profilo basso, quindi possono essere integrati facilmente in dispositivi personalizzati.

Le **Evaluation Boards** sono schede di supporto convenienti per far funzionare i pannelli di visualizzazione Riverdi prima di progettare o acquistare una scheda di supporto per un'applicazione di produzione.

4.10.3 Viewe

Viewe è specializzata in soluzioni di visualizzazione complete da oltre 10 anni, con una notevole capacità di ricerca e sviluppo software e hardware, produzione in fabbrica e assistenza commerciale. Viewe non offre solo soluzioni complete per display e touch ma anche display intelligenti embedded con produzioni standard e personalizzate per trasformare i vostri progetti in realtà.

Portfolio Prodotti:

- Display Intelligenti Embedded
  - IOT_AIOT Smart Display
  - Uart Smart Display
  - HDMI Display_Raspberry Pi Display
  - Display Arduino

Display TFT | OLED

- Sunlight Readable Display
- AMOLED Displays
- Transflective TFT Displays
- Standard TFT Displays
- Bar Display/Stretched TFT Display
- Square TFT Display
- Round TFT Display

IOT AIOT Smart Displays

Gli Smart Display Viewe IOT AIOT sono una serie di display che utilizzano microcontrollori ESP32 con interazioni multimodali integrate di GUI/voce/telecamera, edge computing ML/AI e connettività fluida. Interfacce UART/RS232/RS485/CAN e interfacce IO ridondanti sono incluse per facilitarne l'utilizzo prolungato. LVGL/Arduino/ESP-IDF sono particolarmente adatti a queste schede.

lv_port_viewe_7_espidf

lv_port_viewe_7_espidf è un repository di porte LVGL che supporta lo Smart Display Viewe ESP32 S3 da 7 pollici. Si basa su ESP-IDF e utilizza il microcontrollore ESP32-S3.

- Viewe ESP32 S3 Smart Display

lv_port_viewe_knob_15_espidf

lv_port_viewe_knob_15_espidf è un "port repository" LVGL che supporta il display touch Knob AMOLED Viewe. Si basa su ESP-IDF e utilizza il microcontrollore ESP32-S3.

- Viewe AMOLED Touch Knob Display

Display Intelligenti UART

I display intelligenti UART Viewe sono una serie di display che utilizzano microcontrollori ESP32 e STM32 con interfaccia UART/RS232/RS485/CAN e dispongono di interfaccia IO per la connessione alle periferiche. LVGL è particolarmente adatto a queste schede.

lv_port_viewe_7_espidf

lv_port_viewe_7_espidf è un repository di porte LVGL che supporta lo Smart Display Viewe ESP32 S3 da 7 pollici. Si basa su ESP-IDF e utilizza il microcontrollore ESP32-S3.

- Viewe ESP32 S3 Smart Display

CAPITOLO 5

Common Widget Features

The following details apply to all types of Widgets.

5.1 Widget Basics

5.1.1 What is a Widget?

A Widget is the **basic building block** of the LVGL user interface.

Examples of Widgets: *Base Widget (and Screen)*, *Button*, *Label*, *Image*, *List*, *Chart* and *Text Area*.

See *All Widgets* to see all Widget types.

All Widgets are referenced using an `lv_obj_t` pointer as a handle. This pointer can later be used to read or change the Widget's attributes.

5.1.2 Attributes

Basic attributes

All Widget types share some basic attributes:

- Position
- Size
- Parent
- Stili
- Events it emits
- Flags like *Clickable*, *Scrollable*, etc.
- Ecc.

You can set/get these attributes with `lv_obj_set_...` and `lv_obj_get_...` functions. Ad esempio:

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

For complete details on position, size, coordinates and layouts, see *Positions, Sizes and Layouts*.

Widget-specific attributes

The Widget types have special attributes as well. For example, a slider has

- Minimum and maximum values
- Current value

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

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

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

Parents and children

A Widget's parent is set when the widget is created --- the parent is passed to the creation function.

To get a Widget's current parent, use `lv_obj_get_parent(widget)`.

You can move the Widget to a new parent with `lv_obj_set_parent(widget, new_parent)`.

To get a specific child of a parent use `lv_obj_get_child(parent, idx)`. Some examples for `idx`:

- `0` get the child created first
- `1` get the child created second
- `-1` get the child created last

You can iterate through a parent Widget's children like this:

```
uint32_t i;
for(i = 0; i < lv_obj_get_child_count(parent); i++) {
    lv_obj_t * child = lv_obj_get_child(parent, i);
    /* Do something with child. */
}
```

`lv_obj_get_index(widget)` returns the index of the Widget in its parent. It is equivalent to the number of older children in the parent.

You can bring a Widget to the foreground or send it to the background with `lv_obj_move_foreground(widget)` and `lv_obj_move_background(widget)`.

You can change the index of a Widget in its parent using `lv_obj_move_to_index(widget, index)`.

You can swap the position of two Widgets with `lv_obj_swap(widget1, widget2)`.

To get a Widget's Screen (highest-level parent) use `lv_obj_get_screen(widget)`.

5.1.3 Working Mechanisms

Parent-child structure

A parent Widget can be considered as the container of its children. Every Widget has exactly one parent Widget (except Screens), but a parent Widget can have any number of children. There is no limitation for the type of the parent but there are Widgets which are typically a parent (e.g. button) or a child (e.g. label).

Moving together

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

```
lv_obj_t * parent = lv_obj_create(lv_screen_active());  /* Create a parent␣
↪Widget on current Screen */
lv_obj_set_size(parent, 100, 80);                       /* Set size of␣
↪parent */

lv_obj_t * widget1 = lv_obj_create(parent);             /* Create a Widget␣
↪on previously created parent Widget */
lv_obj_set_pos(widget1, 10, 10);                        /* Set position of␣
↪new Widget */
```

Modify the position of the parent:

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

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

Visibility only on the parent

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

```
lv_obj_set_x(widget1, -30);    /* Move the child a little bit off the␣
↪parent */
```

This behavior can be overwritten with `lv_obj_add_flag(widget, LV_OBJ_FLAG_OVERFLOW_VISIBLE)` which allows the children to be drawn outside of the parent. In addition to this, you must register the following event callback (this was not required in previous versions).

Note: `ext_width` should be the maximum absolute width the children will be drawn within.

```
static void ext_draw_size_event_cb(lv_event_t * e)
{
    lv_event_set_ext_draw_size(e, 30); /* Set 30px extra draw area around␣
↪the widget */
}
```

Creating and deleting Widgets

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

This allows for the creation of a Screen just when a button is clicked to open it, and for deletion of Screens when a new Screen is loaded.

UIs can be created based on the current environment of the device. For example one can create meters, charts, bars and sliders based on the currently attached sensors.

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

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

Typically, the create functions only have a `parent` parameter telling them on which Widget to create the new Widget.

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

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

```
void lv_obj_delete(lv_obj_t * widget);
```

`lv_obj_delete()` will delete the Widget immediately. If for any reason you can't delete the Widget immediately you can use *`lv_obj_delete_async`*`(widget)` which will perform the deletion on the next call of *`lv_timer_handler()`*. This is useful e.g. if you want to delete the parent of a Widget in the child's *`LV_EVENT_DELETE`* handler.

You can remove all the children of a Widget (but not the Widget itself) using *`lv_obj_clean`*`(widget)`.

You can use *`lv_obj_delete_delayed`*`(widget, 1000)` to delete a Widget after some time. The delay is expressed in milliseconds.

Sometimes you're not sure whether a Widget was deleted and you need some way to check if it's still "alive". Anytime before the Widget is deleted, you can use cpp:expr:`lv_obj_null_on_delete(&widget)` to cause your Widget pointer to be set to `NULL` when the Widget is deleted.

Make sure the pointer variable itself stays valid until the Widget is deleted. Here is an example:

```
void some_timer_callback(lv_timer_t * t)
{
   static lv_obj_t * my_label;
   if(my_label == NULL) {
      my_label = lv_label_create(lv_screen_active());
      lv_obj_delete_delayed(my_label, 1000);
      lv_obj_null_on_delete(&my_label);
   }
   else {
      lv_obj_set_x(my_label, lv_obj_get_x(my_label) + 1);
   }
}
```

5.1.4 Screen

What are Screens?

Not to be confused with a *Display (lv_display)*, Screens are simply any Widget created without a parent (i.e. passing NULL for the `parent` argument during creation). As such, they form the "root" of a Widget Tree.

Normally the Base Widget is used for this purpose since it has all the features most Screens need. But an *Image (lv_image)* Widget can also be used to create a wallpaper background for the Widget Tree.

All Screens:

- are automatically attached to the *Display di Default* current when the Screen was created;
- automatically occupy the full area of the associated display;
- cannot be moved, i.e. functions such as *`lv_obj_set_pos()`* and *`lv_obj_set_size()`* cannot be used on screens.

Each *Display (lv_display)* object can have multiple screens associated with it, but not vice versa. Thus the relationship:

```
   Display
      |
     --- (one or more)
     /|\
Screen Widgets  (root of a Widget Tree)
```

(continues on next page)

(continua dalla pagina precedente)

```
      |
      O  (zero or more)
     /|\
Child Widgets
```

Creating Screens

Screens are created like this:

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

Screens can be deleted with `lv_obj_delete(scr)`, but be sure you do not delete the *Lo Screen Attivo*.

Lo Screen Attivo

While each *Display (lv_display)* object can have any number of Screens Widgets associated with it, only one of those Screens is considered "Active" at any given time. That Screen is referred to as the Display's "Active Screen". For this reason, only one Screen and its child Widgets will ever be shown on a display at one time.

When each *Display (lv_display)* object was created, a default screen was created with it and set as its "Active Screen".

To get a pointer to the "Active Screen", call `lv_screen_active()`.

To set a Screen to be the "Active Screen", call `lv_screen_load()` or `lv_screen_load_anim()`.

Loading Screens

To load a new screen, use `lv_screen_load(scr1)`. This sets `scr1` as the Active Screen.

Load Screen with Extended Options

There is a way to load screens that gives you 2 additional (extended) options, allowing the caller to specify:

- an optional transition method, and
- an option to gracefully delete the screen that was being displayed.

`lv_screen_load_anim(scr, transition_type, time, delay, auto_del)`. The following transition types exist:

- `LV_SCREEN_LOAD_ANIM_NONE`: Switch immediately after `delay` milliseconds
- `LV_SCREEN_LOAD_ANIM_OVER_LEFT`, `LV_SCREEN_LOAD_ANIM_OVER_RIGHT`, `LV_SCREEN_LOAD_ANIM_OVER_TOP` and `LV_SCREEN_LOAD_ANIM_OVER_BOTTOM`: Move the new screen over the current towards the given direction
- `LV_SCREEN_LOAD_ANIM_OUT_LEFT`, `LV_SCREEN_LOAD_ANIM_OUT_RIGHT`, `LV_SCREEN_LOAD_ANIM_OUT_TOP` and `LV_SCREEN_LOAD_ANIM_OUT_BOTTOM`: Move out the old screen over the current towards the given direction
- `LV_SCREEN_LOAD_ANIM_MOVE_LEFT`, `LV_SCREEN_LOAD_ANIM_MOVE_RIGHT`, `LV_SCREEN_LOAD_ANIM_MOVE_TOP` and `LV_SCREEN_LOAD_ANIM_MOVE_BOTTOM`: Move both the current and new screens towards the given direction
- `LV_SCREEN_LOAD_ANIM_FADE_IN` and `LV_SCREEN_LOAD_ANIM_FADE_OUT`: Fade the new screen over the old screen, or vice versa

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

The new screen will become active (returned by `lv_screen_active()`) when the animation starts after `delay` time. All inputs are disabled during the screen animation.

I Layer

When an `lv_display_t` object is created, 4 Screens (layers) are created and attached to it.

1. Bottom Layer
2. Lo Screen Attivo
3. Top Layer
4. System Layer

1, 3 and 4 are independent of the *Lo Screen Attivo* and they will be shown (if they contain anything that is visible) regardless of which screen is the Active Screen. See *Screen Layers* and *Transparent Screens* for more information.

5.1.5 Parti

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

The following predefined parts exist in LVGL:

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

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

5.1.6 Stati

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

- `LV_STATE_DEFAULT`: Normale, stato rilasciato
- `LV_STATE_CHECKED`: Stato attivato o selezionato
- `LV_STATE_FOCUSED`: Focus tramite tastiera o encoder o clic tramite touchpad/mouse
- `LV_STATE_FOCUS_KEY`: Focus tramite tastiera o encoder ma non tramite touchpad/mouse
- `LV_STATE_EDITED`: Edit by an encoder
- `LV_STATE_HOVERED`: Hovered by mouse (not supported now)
- `LV_STATE_PRESSED`: Premuto
- `LV_STATE_SCROLLED`: In corso di scrolling
- `LV_STATE_DISABLED`: Disabled state
- `LV_STATE_USER_1`: Custom state
- `LV_STATE_USER_2`: Custom state
- `LV_STATE_USER_3`: Custom state
- `LV_STATE_USER_4`: Custom state

The states are usually automatically changed by the library as the user interacts with a Widget (presses, releases, focuses, etc.). However, the states can be changed manually as well. To set or clear given state (but leave the other states untouched) use `lv_obj_add_state(widget, LV_STATE_...)` and `lv_obj_remove_state(widget, LV_STATE_...)`. In both cases OR-ed state values can be used as well. E.g. `lv_obj_add_state(widget, LV_STATE_PRESSED | LV_PRESSED_CHECKED)`.

To learn more about the states, read the related section of *Styles Overview*.

5.1.7 Flags

There are some Widget attributes which can be enabled/disabled by `lv_obj_add_flag(widget, LV_OBJ_FLAG_...)` and `lv_obj_remove_flag(widget, LV_OBJ_FLAG_...)`.

- `LV_OBJ_FLAG_HIDDEN` Make the Widget hidden. (Like it wasn't there at all)
- `LV_OBJ_FLAG_CLICKABLE` Make the Widget clickable by input devices
- `LV_OBJ_FLAG_CLICK_FOCUSABLE` Add focused state to the Widget when clicked
- `LV_OBJ_FLAG_CHECKABLE` Toggle checked state when the Widget is clicked
- `LV_OBJ_FLAG_SCROLLABLE` Make the Widget scrollable
- `LV_OBJ_FLAG_SCROLL_ELASTIC` Allow scrolling inside but with slower speed
- `LV_OBJ_FLAG_SCROLL_MOMENTUM` Make the Widget scroll further when "thrown"
- `LV_OBJ_FLAG_SCROLL_ONE` Allow scrolling only one snappable children
- `LV_OBJ_FLAG_SCROLL_CHAIN_HOR` Allow propagating the horizontal scroll to a parent
- `LV_OBJ_FLAG_SCROLL_CHAIN_VER` Allow propagating the vertical scroll to a parent
- `LV_OBJ_FLAG_SCROLL_CHAIN` Simple packaging for (`LV_OBJ_FLAG_SCROLL_CHAIN_HOR | LV_OBJ_FLAG_SCROLL_CHAIN_VER`)
- `LV_OBJ_FLAG_SCROLL_ON_FOCUS` Automatically scroll Widget to make it visible when focused
- `LV_OBJ_FLAG_SCROLL_WITH_ARROW` Allow scrolling the focused Widget with arrow keys
- `LV_OBJ_FLAG_SNAPPABLE` If scroll snap is enabled on the parent it can snap to this Widget
- `LV_OBJ_FLAG_PRESS_LOCK` Keep the Widget pressed even if the press slid from the Widget
- `LV_OBJ_FLAG_EVENT_BUBBLE` Propagate the events to the parent as well
- `LV_OBJ_FLAG_EVENT_TRICKLE` Propagate the events to the children as well
- `LV_OBJ_FLAG_STATE_TRICKLE` Propagate the state changes to the children as well
- `LV_OBJ_FLAG_GESTURE_BUBBLE` Propagate the gestures to the parent
- `LV_OBJ_FLAG_ADV_HITTEST` Allow performing more accurate hit (click) test. E.g. accounting for rounded corners
- `LV_OBJ_FLAG_IGNORE_LAYOUT` Make the Widget not positioned by the layouts
- `LV_OBJ_FLAG_FLOATING` Do not scroll the Widget when the parent scrolls and ignore layout
- `LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS` Enable sending `LV_EVENT_DRAW_TASK_ADDED` events
- `LV_OBJ_FLAG_OVERFLOW_VISIBLE` Do not clip the children's content to the parent's boundary
- `LV_OBJ_FLAG_FLEX_IN_NEW_TRACK` Start a new flex track on this item
- `LV_OBJ_FLAG_LAYOUT_1` Custom flag, free to use by layouts
- `LV_OBJ_FLAG_LAYOUT_2` Custom flag, free to use by layouts
- `LV_OBJ_FLAG_WIDGET_1` Custom flag, free to use by widget
- `LV_OBJ_FLAG_WIDGET_2` Custom flag, free to use by widget

- *LV_OBJ_FLAG_USER_1* Custom flag, free to use by user
- *LV_OBJ_FLAG_USER_2* Custom flag, free to use by user
- *LV_OBJ_FLAG_USER_3* Custom flag, free to use by user
- *LV_OBJ_FLAG_USER_4* Custom flag, free to use by user

Alcuni esempi:

```
/* Hide on Widget */
lv_obj_add_flag(widget, LV_OBJ_FLAG_HIDDEN);

/* Make a Widget non-clickable */
lv_obj_remove_flag(widget, LV_OBJ_FLAG_CLICKABLE);
```

5.1.8 Base-Widget Events

Events from Input Devices

- *LV_EVENT_PRESSED* Widget has been pressed.
- *LV_EVENT_PRESSING* Widget is being pressed (sent continuously while pressing).
- *LV_EVENT_PRESS_LOST* Widget is still being pressed but slid cursor/finger off Widget.
- *LV_EVENT_SHORT_CLICKED* Widget was pressed for a short period of time, then released. Not sent if scrolled.
- *LV_EVENT_SINGLE_CLICKED* Sent for first short click within a small distance and short time.
- *LV_EVENT_DOUBLE_CLICKED* Sent for second short click within small distance and short time.
- *LV_EVENT_TRIPLE_CLICKED* Sent for third short click within small distance and short time.
- *LV_EVENT_LONG_PRESSED* Object has been pressed for at least `long_press_time`. Not sent if scrolled.
- *LV_EVENT_LONG_PRESSED_REPEAT* Sent after `long_press_time` in every `long_press_repeat_time` ms. Not sent if scrolled.
- *LV_EVENT_CLICKED* Sent on release if not scrolled (regardless to long press).
- *LV_EVENT_RELEASED* Sent in every cases when Widget has been released.
- *LV_EVENT_SCROLL_BEGIN* Scrolling begins. The event parameter is a pointer to the animation of the scroll. Can be modified.
- *LV_EVENT_SCROLL_THROW_BEGIN* Received when scrolling begins.
- *LV_EVENT_SCROLL_END* Scrolling ended.
- *LV_EVENT_SCROLL* Scrolling
- *LV_EVENT_GESTURE* A gesture is detected. Get gesture with `lv_indev_get_gesture_dir(lv_indev_active());`
- *LV_EVENT_KEY* A key is sent to Widget. Get key with `lv_indev_get_key(lv_indev_active());`
- *LV_EVENT_FOCUSED* Widget received focus,
- *LV_EVENT_DEFOCUSED* Widget's focus has been lost.
- *LV_EVENT_LEAVE* Widget's focus has been lost but is still selected.
- *LV_EVENT_HIT_TEST* Perform advanced hit-testing.

Special Events

- `LV_EVENT_VALUE_CHANGED` when the `LV_OBJ_FLAG_CHECKABLE` flag is enabled and the Widget was clicked (on transition to/from the checked state)

Drawing Events

- `LV_EVENT_DRAW_MAIN` Performing drawing of main part
- `LV_EVENT_DRAW_MAIN_BEGIN` Starting drawing of main part
- `LV_EVENT_DRAW_MAIN_END` Finishing drawing of main part
- `LV_EVENT_DRAW_POST` Perform the post draw phase (when all children are drawn)
- `LV_EVENT_DRAW_POST_BEGIN` Starting the post draw phase (when all children are drawn)
- `LV_EVENT_DRAW_POST_END` Finishing the post draw phase (when all children are drawn)

Other Events

- `LV_EVENT_DELETE` Object is being deleted
- `LV_EVENT_CHILD_CHANGED` Child was removed, added, or its size, position were changed
- `LV_EVENT_CHILD_CREATED` Child was created, always bubbles up to all parents
- `LV_EVENT_CHILD_DELETED` Child was deleted, always bubbles up to all parents
- `LV_EVENT_SIZE_CHANGED` Object coordinates/size have changed
- `LV_EVENT_STYLE_CHANGED` Object's style has changed
- `LV_EVENT_LAYOUT_CHANGED` A child's position has changed due to a layout recalculation (when container has flex or grid layout style)
- `LV_EVENT_GET_SELF_SIZE` Get internal size of a widget

Further Reading

Learn more about *Eventi*.

5.1.9 Keys

If `LV_OBJ_FLAG_CHECKABLE` is enabled, `LV_KEY_RIGHT` and `LV_KEY_UP` make the Widget checked, and `LV_KEY_LEFT` and `LV_KEY_DOWN` make it unchecked.

If `LV_OBJ_FLAG_SCROLLABLE` is enabled, but the Widget is not editable (as declared by the widget class), the arrow keys (`LV_KEY_UP`, `LV_KEY_DOWN`, `LV_KEY_LEFT`, `LV_KEY_RIGHT`) scroll the Widget. If the Widget can only scroll vertically, `LV_KEY_LEFT` and `LV_KEY_RIGHT` will scroll up/down instead, making it compatible with an encoder input device. See *Input devices overview* for more on encoder behaviors and the edit mode.

Further Reading

Learn more about *Keys*.

5.1.10 Names

When a widget is created, its reference can be stored in an `lv_obj_t*` pointer variable. To use this widget in multiple places in the code, the variable can be passed as a function parameter or made a global variable. However, this approach has some drawbacks:

- Using global variables is not clean and generally not recommended.
- It's not scalable. Passing references to 20 widgets as function parameters is not ideal.

- It's hard to track whether a widget still exists or has been deleted.

Setting names

To address these issues, LVGL introduces a powerful widget naming system that can be enabled by setting `LV_USE_OBJ_NAME` in `lv_conf.h`.

A custom name can be assigned using `lv_obj_set_name(obj, "name")` or `lv_obj_set_name_static(obj, "name")`. The "static" variant means the passed name must remain valid while the widget exists, as only the pointer is stored. Otherwise, LVGL will allocate memory to store a copy of the name.

If a name ends with `#`, LVGL will automatically replace it with an index based on the number of siblings with the same base name. If no name is provided, the default is `<widget_type>_#`.

Below is an example showing how manually- and automatically-assigned names are resolved:

- Main `lv_obj` container named `"cont"`: "cont"
  - `lv_obj` container named `"header"`: "header"
    - `lv_label` with no name: "lv_label_0"
    - `lv_label` named `"title"`: "title"
    - `lv_label` with no name: "lv_label_1" (It's the third label, but custom-named widgets are not counted)
  - `lv_obj` container named `"buttons"`:
    - `lv_button` with no name: "lv_button_0"
    - `lv_button` named `"second_button"`: "second_button"
    - `lv_button` with no name: "lv_button_1"
    - `lv_button` named `lv_button_#`: "lv_button_2"
    - `lv_button` named `mybtn_#`: "mybtn_0"
    - `lv_button` with no name: "lv_button_2"
    - `lv_button` named `mybtn_#`: "mybtn_1"
    - `lv_button` named `mybtn_#`: "mybtn_2"
    - `lv_button` named `mybtn_#`: "mybtn_3"

Finding widgets

Widgets can be found by name in two ways:

1. **Get a direct child by name** using `lv_obj_get_child_by_name(parent, "child_name")`. Example: `lv_obj_get_child_by_name(header, "title")` You can also use a "path" to find nested children: `lv_obj_get_child_by_name(cont, "buttons/mybtn_2")`
2. **Find a descendant at any level** using `lv_obj_find_by_name(parent, "child_name")`. Example: `lv_obj_find_by_name(cont, "mybtn_1")` Note that `"mybtn_1"` is a child of `buttons`, not a direct child of `cont`. This is useful when you want to ignore hierarchy and search by name alone.

Since both functions start searching from a specific parent, it's possible to have multiple widget subtrees with identical names under different parents.

For example, if `my_listitem_create(parent)` creates a widget named `"list_item_#"` with children like `"icon"`, `"title"`, `"ok_button"`, and `"lv_label_0"`, and it's called 10 times, a specific `"ok_button"` can be found like this:

```
lv_obj_t * item = lv_obj_find_by_name(lv_screen_active(), "list_item_5");
lv_obj_t * ok_btn = lv_obj_find_by_name(item, "ok_button");

// Or
lv_obj_t * ok_btn = lv_obj_get_child_by_name(some_list_container, "list_
↪item_5/ok_button");
```

Names are resolved **when they are retrieved**, not when they are set. This means the indices always reflect the current state of the widget tree at the time the name is used.

5.1.11 Snapshot

A snapshot image can be generated for a Widget together with its children. Check details in *Snapshot*.

5.1.12 Esempio

Oggetti base con stili personalizzati

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

void lv_example_obj_1(void)
{
    lv_obj_t * obj1;
    obj1 = lv_obj_create(lv_screen_active());
    lv_obj_set_size(obj1, 100, 50);
    lv_obj_align(obj1, LV_ALIGN_CENTER, -60, -30);

    static lv_style_t style_shadow;
    lv_style_init(&style_shadow);
    lv_style_set_shadow_width(&style_shadow, 10);
    lv_style_set_shadow_spread(&style_shadow, 5);
    lv_style_set_shadow_color(&style_shadow, lv_palette_main(LV_PALETTE_
↪BLUE));

    lv_obj_t * obj2;
    obj2 = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj2, &style_shadow, 0);
    lv_obj_align(obj2, LV_ALIGN_CENTER, 60, 30);
}
#endif
```

Creare un oggetto trascinabile

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

static void drag_event_handler(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);

    lv_indev_t * indev = lv_indev_active();
    if(indev == NULL)  return;

    lv_point_t vect;
    lv_indev_get_vect(indev, &vect);
```

(continues on next page)

(continua dalla pagina precedente)

```
    int32_t x = lv_obj_get_x_aligned(obj) + vect.x;
    int32_t y = lv_obj_get_y_aligned(obj) + vect.y;
    lv_obj_set_pos(obj, x, y);
}

/**
 * Make an object draggable.
 */
void lv_example_obj_2(void)
{
    lv_obj_t * obj;
    obj = lv_obj_create(lv_screen_active());
    lv_obj_set_size(obj, 150, 100);
    lv_obj_add_event_cb(obj, drag_event_handler, LV_EVENT_PRESSING, NULL);

    lv_obj_t * label = lv_label_create(obj);
    lv_label_set_text(label, "Drag me");
    lv_obj_center(label);

}
#endif
```

Trasformare un oggetto usando una matrice 3x3

```
#include "../../lv_examples.h"
#if LV_BUILD_EXAMPLES
#if LV_DRAW_TRANSFORM_USE_MATRIX

static void timer_cb(lv_timer_t * timer)
{
    lv_obj_t * obj = (lv_obj_t *) lv_timer_get_user_data(timer);

    static float value = 0.1f;
    lv_matrix_t matrix;
    lv_matrix_identity(&matrix);
    lv_matrix_scale(&matrix, value, 1);
    lv_matrix_rotate(&matrix, value * 360);
    lv_obj_set_transform(obj, &matrix);

    value += 0.01f;

    if(value > 2.0f) {
        lv_obj_reset_transform(obj);
        value = 0.1f;
    }
}

void lv_example_obj_3(void)
{
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_center(obj);

    lv_timer_create(timer_cb, 20, obj);
}
```

(continues on next page)

(continua dalla pagina precedente)

```
#else

void lv_example_obj_3(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text_static(label, "LV_DRAW_TRANSFORM_USE_MATRIX is not␣
↪enabled");
    lv_obj_center(label);
}

#endif /*LV_DRAW_TRANSFORM_USE_MATRIX*/
#endif /*LV_BUILD_EXAMPLES*/
```

5.1.13 API

lv_api_map_v8.h

lv_api_map_v9_1.h

lv_display.h

lv_flex.h

lv_grid.h

lv_obj.h

lv_obj_class.h

lv_obj_class_private.h

lv_obj_draw.h

lv_obj_draw_private.h

lv_obj_event.h

lv_obj_pos.h

lv_obj_private.h

lv_obj_property_names.h

lv_obj_scroll.h

lv_obj_scroll_private.h

lv_obj_style.h

lv_obj_style_gen.h

lv_obj_style_private.h

lv_obj_tree.h

lv_observer.h

lv_refr.h

lv_refr_private.h

lv_xml_obj_parser.h

5.2 Positions, Sizes and Layouts

5.2.1 Panoramica

Similar to many other parts of LVGL, the concept of setting the coordinates was inspired by CSS. LVGL has by no means a complete implementation of CSS but a comparable subset is implemented (sometimes with minor adjustments).

In short this means:

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

5.2.2 Length Units

When passing "length units" (a.k.a. "distance units" or "size units") as arguments to functions that modify position, size, etc., to make layout of your UI convenient, you have a choice of several different types of units you can use.

pixels
Specify size as pixels: an integer value < *LV_COORD_MAX* always means pixels. E.g. *lv_obj_set_x*(btn, 10).

percentage
Specify size as a percentage of the size of the Widget's parent or of itself, depending on the property. *lv_pct*(value) converts value to a percentage. E.g. *lv_obj_set_width*(btn, *lv_pct*(50)). If you want to avoid the overhead of the call to *lv_pct()*, you can also use the macro LV_PCT(x) to mean the same thing. Note that when you use this feature, your value is *stored as a percent* so that if/when the size of the parent container (or other positioning factor) changes, this style value dynamically retains its meaning.

contained content
Specify size as a function of the Widget's children. The macro *LV_SIZE_CONTENT*: passed as a size value has special meaning: it means to set the width and/or height of a Widget just large enough to include all of its children. This is similar to auto in CSS. E.g. *lv_obj_set_width*(btn, LV_SIZE_CONTENT).

inches
Specify size as 1/160-th portion of an inch as if it were pixels on a 160-DPI display, even though a display may have a different DPI. Use *lv_dpx*(n) or LV_DPX(n) to do this. Examples:

n	DPI	Computed Pixels
40	320	80 pixels to make 1/4 inch
40	160	40 pixels to make 1/4 inch
40	130	33 pixels to make 1/4 inch
80	130	66 pixels to make 1/2 inch

See DPI under *Attributes*.

5.2.3 Boxing Model

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

bounding box
the width/height of the elements.

border width
the width of the border.

padding
space between the sides of the Widget and its children.

margin
space outside of the Widget (considered only by some layouts)

content
the content area which is the size of the bounding box reduced by the border width and padding.

The border is drawn inside the bounding box. Inside the border LVGL keeps a "padding margin" when placing a Widget's children.

The outline is drawn outside the bounding box.

5.2.4 Important Notes

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

Postponed coordinate calculation

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

In other words, if you need to get the coordinate of a Widget and the coordinates were just changed, LVGL needs to be forced to recalculate the coordinates. To do this call `lv_obj_update_layout()`.

The size and position might depend on the parent or layout. Therefore `lv_obj_update_layout()` recalculates the coordinates of all Widgets on the screen of `obj`.

Removing styles

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

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

```
lv_obj_remove_style_all(widget)
```

or

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

the earlier set coordinates will be removed as well.

Ad esempio:

```
/* The size of obj1 will be set back to the default in the end */
lv_obj_set_size(widget1, 200, 100);  /* Now obj1 has 200;100 size */
lv_obj_remove_style_all(widget1);    /* It removes the set sizes */


/* widget2 will have 200;100 size in the end */
lv_obj_remove_style_all(widget2);
lv_obj_set_size(widget2, 200, 100);
```

5.2.5 Positioning Widgets

Direct

To simply set the x and y coordinates of a Widget use:

```
lv_obj_set_x(widget, 10);          /* Separate... */
lv_obj_set_y(widget, 20);
lv_obj_set_pos(widget, 10, 20);    /* Or in one function */
```

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

Percentage values are calculated from the parent's content area size.

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

Alignment

Inside parent widget

In many cases it is more convenient to tell LVGL to align your object relative to an "anchor" in its parent *other* than its upper left corner. To establish that "anchor", call *`lv_obj_set_align`*`(widget, LV_ALIGN_...)`. After that call, that "anchor" will be remembered until another one is established. In other words, every futire x and y setting for that Widget will be relative to the that "anchor".

Example: Position Widget (10,20) px relative to the center of its parent:

```
lv_obj_set_align(widget, LV_ALIGN_CENTER);
lv_obj_set_pos(widget, 10, 20);

/* Or combine the above in one function... */
lv_obj_align(widget, LV_ALIGN_CENTER, 10, 20);
```

9 convenient "anchors" can be used with these functions:

- *`LV_ALIGN_TOP_LEFT`*
- *`LV_ALIGN_TOP_MID`*
- *`LV_ALIGN_TOP_RIGHT`*
- *`LV_ALIGN_BOTTOM_LEFT`*
- *`LV_ALIGN_BOTTOM_MID`*
- *`LV_ALIGN_BOTTOM_RIGHT`*

- *LV_ALIGN_LEFT_MID*
- *LV_ALIGN_RIGHT_MID*
- *LV_ALIGN_CENTER*

See illustration below to visualize what these mean.

It's quite common to align a child to the center of its parent, therefore a dedicated function exists:

```
lv_obj_center(widget);

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

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

Relative to another Widget

Alternately, you can choose an "anchor" on another Widget.

```
lv_obj_align_to(widget, reference_widget, align, x, y);
```

where `align` can be done of the following:

- *LV_ALIGN_OUT_TOP_LEFT*
- *LV_ALIGN_OUT_TOP_MID*
- *LV_ALIGN_OUT_TOP_RIGHT*
- *LV_ALIGN_OUT_BOTTOM_LEFT*
- *LV_ALIGN_OUT_BOTTOM_MID*
- *LV_ALIGN_OUT_BOTTOM_RIGHT*
- *LV_ALIGN_OUT_LEFT_TOP*
- *LV_ALIGN_OUT_LEFT_MID*
- *LV_ALIGN_OUT_LEFT_BOTTOM*
- *LV_ALIGN_OUT_RIGHT_TOP*
- *LV_ALIGN_OUT_RIGHT_MID*
- *LV_ALIGN_OUT_RIGHT_BOTTOM*

Example: to horizontally center a label 10 pixels above a button:

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

Note that, unlike with *lv_obj_align()*, *lv_obj_align_to()* does not remember the "anchor" used, and so will not automatically reposition the aligned widget if the reference widget later moves.

The following illustration shows the meaning of each "anchor" mentioned above.

5.2.6 Size

Sizing the simple way

The width and the height of a Widget can be set easily as well:

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

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

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

The size settings support a special value: *LV_SIZE_CONTENT*. It means the Widget's size in the respective direction will be set to the size of its children. Note that only children on the right and bottom sides will be considered and children on the top and left remain cropped. This limitation makes the behavior more predictable.

Widgets with *LV_OBJ_FLAG_HIDDEN* or *LV_OBJ_FLAG_FLOATING* will be ignored by the *LV_SIZE_CONTENT* calculation.

The above functions set the size of a Widget's bounding box but the size of the content area can be set as well. This means a Widget's bounding box will be enlarged with the addition of padding.

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

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

```
int32_t w = lv_obj_get_width(widget);
int32_t h = lv_obj_get_height(widget);
int32_t content_w = lv_obj_get_content_width(widget);
int32_t content_h = lv_obj_get_content_height(widget);
```

Extending the click area

By default, Widgets can be clicked only within their bounding area. However, especially with small Widgets, it can be helpful to make a Widget's "clickable" area larger. You can do this with `lv_obj_set_ext_click_area(widget, size)`.

5.2.7 Using styles

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

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

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

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

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

lv_obj_t * btn = lv_button_create(lv_screen_active());
lv_obj_add_style(btn, &style, LV_PART_MAIN);
```

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

5.2.8 Translation

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

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

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

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

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

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

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

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

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

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

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

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

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

Translation is applied from the current position of the Widget.

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

The translation is applied after the layouts are calculated. Therefore, even laid out Widgets' position can be translated.

The translation actually moves the Widget. That means it makes the scrollbars and `LV_SIZE_CONTENT` sized Widgets react to the position change.

5.2.9 Transformation

Similarly to position, a Widget's size can be changed relative to the current size as well. The transformed width and height are added on both sides of the Widget. This means a 10 px transformed width makes the Widget 2x10 pixels wider.

Unlike position translation, the size transformation doesn't make the Widget "really" larger. In other words scrollbars, layouts, and `LV_SIZE_CONTENT` will not react to the transformed size. Hence, size transformation is "only" a visual effect.

This code enlarges a button when it's pressed:

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

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

Min and Max size

Similarly to CSS, LVGL also supports `min-width`, `max-width`, `min-height` and `max-height`. These are limits preventing a Widget's size from becoming smaller/larger than these values. They are especially useful if the size is set by percentage or `LV_SIZE_CONTENT`.

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

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

(continues on next page)

(continua dalla pagina precedente)

```
lv_obj_add_style(widget, &style_max_height, LV_STATE_DEFAULT); //Limit the␣
↪ height to 200 px
```

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

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

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

5.2.10 Layout

Layout overview

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

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

There is only one function that is the same for every layout: *lv_obj_set_layout()* `(widget, <LAYOUT_NAME>)` sets the layout on a Widget. For further settings of the parent and children see the documentation of the given layout.

Built-in layouts

LVGL comes with two very powerful layouts:

- Flexbox: arrange Widgets into rows or columns, with support for wrapping and expanding items.
- Grid: arrange Widgets into fixed positions in 2D table.

Both are heavily inspired by the CSS layouts with the same name. Layouts are described in detail in their own section of documentation.

Flags

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

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

These flags can be added/removed with *lv_obj_add_flag*`(widget, FLAG)` and *lv_obj_remove_flag*`(widget, FLAG)`

Adding new layouts

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

```
uint32_t MY_LAYOUT;

...

MY_LAYOUT = lv_layout_register(my_layout_update, &user_data);
```

(continues on next page)

(continua dalla pagina precedente)

```
...

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

Custom style properties can be added which can be retrieved and used in the update callback. Ad esempio:

```
uint32_t MY_PROP;
...

LV_STYLE_MY_PROP = lv_style_register_prop();

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

5.2.11 Esempi

5.2.12 API

5.3 I Layer

When the term "layer" is used in LVGL documentation, it may refer to one of several things:

1. for Widgets, the *Order of Creation* creates a natural layering of Widgets;
2. in the context of pixel rendering (drawing), there are *Draw Layers*;
3. permanent *Screen Layers* are part of each *Display (lv_display)* object.

#1 is covered below. #2 and #3 are covered in *Draw Layers* and *Screen Layers* respectively.

5.3.1 Order of Creation

By default, LVGL draws new Widgets on top of old Widgets.

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

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

/* Create 2 buttons */
lv_obj_t * btn1 = lv_button_create(scr);     /* Create the first button on␣
↪the screen */
lv_obj_set_pos(btn1, 60, 40);                      /* Set the position of␣
↪the first button */

lv_obj_t * btn2 = lv_button_create(scr);     /* Create the second button␣
↪on the screen */
lv_obj_set_pos(btn2, 180, 80);                     /* Set the position of␣
↪the second button */

/* Add labels to the buttons */
lv_obj_t * label1 = lv_label_create(btn1);   /* Create a label on the␣
↪first button */
lv_label_set_text(label1, "Button 1");             /* Set the text of the␣
↪label */

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

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

Changing Order

There are four explicit ways to bring a Widget to the foreground:

- Use `lv_obj_move_foreground(widget)` to bring a Widget to the foreground. Similarly, use `lv_obj_move_background(widget)` to move it to the background.
- Use `lv_obj_move_to_index(widget, idx)` to move a Widget to a given index in the order of children.

  - `0`: background
  - `child_num - 1`: foreground
  - `< 0`: count from the top, to move forward (up): `lv_obj_move_to_index(widget, lv_obj_get_index(widget) - 1)`
- Use `lv_obj_swap(widget1, widget2)` to swap the relative layer position of two Widgets.
- When `lv_obj_set_parent(widget, new_parent)` is used, `widget` will be on the foreground of `new_parent`.

5.3.2 API

lv_api_map_v8.h

lv_obj_tree.h

5.4 Stili

5.4.1 Styles Overview

Styles are used to set the appearance of Widgets. Styles in LVGL are heavily inspired by CSS. The concept in a nutshell is that a style is an `lv_style_t` variable which can hold properties like border width, font, text color and so on. It's similar to a `class` in CSS.

- Styles can be assigned to Widgets to change their appearance. Upon assignment, the target part (pseudo-element in CSS) and target state (pseudo-class in CSS) can be specified. For example one can add `style_blue` to the knob of a slider when it's in pressed state.
- The same style can be used by any number of Widgets.
- Styles can be cascaded which means multiple styles may be assigned to a Widget and each style can have different properties. Therefore, not all properties have to be specified in a style. LVGL will search for a property until a style defines it or use a default value if it's not specified by any of the styles. For example `style_btn` can result in a default gray button and `style_btn_red` can add only a `background-color=red` to overwrite the background color.
- The most recently added style has higher precedence. This means if a property is specified in two styles the newest style in the Widget will be used.
- Some properties (e.g. text color) can be inherited from a parent(s) if it's not specified in a Widget.
- Widgets can also have *local styles* with higher precedence than "normal" styles.
- Unlike CSS (where pseudo-classes describe different states, e.g. `:focus`), in LVGL a property is assigned to a given state.
- Transitions can be applied when the Widget changes state.

Stati

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

- `LV_STATE_DEFAULT`: (0x0000) Normal, released state
- `LV_STATE_CHECKED`: (0x0001) Toggled or checked state
- `LV_STATE_FOCUSED`: (0x0002) Focused via keypad or encoder or clicked via touchpad/mouse
- `LV_STATE_FOCUS_KEY`: (0x0004) Focused via keypad or encoder but not via touchpad/mouse
- `LV_STATE_EDITED`: (0x0008) Edit by an encoder
- `LV_STATE_HOVERED`: (0x0010) Hovered by mouse
- `LV_STATE_PRESSED`: (0x0020) Being pressed

- *LV_STATE_SCROLLED*: (0x0040) Being scrolled
- *LV_STATE_DISABLED*: (0x0080) Disabled state
- *LV_STATE_USER_1*: (0x1000) Custom state
- *LV_STATE_USER_2*: (0x2000) Custom state
- *LV_STATE_USER_3*: (0x4000) Custom state
- *LV_STATE_USER_4*: (0x8000) Custom state

A Widget can be in a combination of states such as being focused and pressed at the same time. This is represented as *LV_STATE_FOCUSED* | *LV_STATE_PRESSED*.

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

Since *LV_PART_MAIN* and *LV_STATE_DEFAULT* both have zero values, you can simply pass `0` as the `selector` argument instead of `LV_PART_MAIN | LV_STATE_DEFAULT` as a shortcut when adding styles to an object.

What does the "best matching state's property" mean?

States have a precedence which is shown by their value (see in the above list). A higher value means higher precedence. To determine which state's property to use let's take an example. Imagine the background color is defined like this:

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

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

Some practical notes:

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

etc. states is quite difficult. Instead, for example, use the background color for pressed and checked states and indicate the focused state with a different border color.

Cascading Styles

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

This is much like in CSS when used classes are listed like `<div class=".btn .btn-red">`.

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

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

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

Ereditarietà

Some properties (typically those related to text) can be inherited from the parent Widget's styles. Inheritance is applied only if the given property is not set in the Widget's styles (even in default state). In this case, if the property is inheritable, the property's value will be searched up the parent hierarchy until a Widget specifies a value for the property. The parents will use their own state to determine the value. So if a button is pressed, and the text color comes from a parent, the pressed text color will be used.

Parti

Widgets can be composed of *parts* which may each have their own styles.

The following predefined parts exist in LVGL:

- *LV_PART_MAIN*: (0x000000) A background like rectangle
- *LV_PART_SCROLLBAR*: (0x010000) The scrollbar(s)
- *LV_PART_INDICATOR*: (0x020000) Indicator, e.g. for slider, bar, switch, or the tick box of the checkbox
- *LV_PART_KNOB*: (0x030000) Like a handle to grab to adjust a value
- *LV_PART_SELECTED*: (0x040000) Indicate the currently selected option or section
- *LV_PART_ITEMS*: (0x050000) Used if the widget has multiple similar elements (e.g. table cells)
- *LV_PART_CURSOR*: (0x060000) Mark a specific place e.g. Text Area's or chart's cursor
- *LV_PART_CUSTOM_FIRST*: (0x080000) Custom part identifiers can be added starting from here.
- *LV_PART_ANY*: (0x0F0000) Special value can be used in some functions to target all parts.

For example a *Slider* has three parts:

- Main (background)
- Indicator
- Knob

This means all three parts of the slider can have their own styles. See later how to add styles to Widgets and parts.

Since *LV_PART_MAIN* and *LV_STATE_DEFAULT* both have zero values, you can simply pass `0` as the `selector` argument instead of `LV_PART_MAIN | LV_STATE_DEFAULT` as a shortcut when adding styles to an object.

Initialize Styles and Set/Get Properties

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

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

```
static lv_style_t style_btn;
lv_style_init(&style_btn);
lv_style_set_bg_color(&style_btn, lv_color_hex(0x115588));
lv_style_set_bg_opa(&style_btn, LV_OPA_50);
lv_style_set_border_width(&style_btn, 2);
lv_style_set_border_color(&style_btn, lv_color_black());

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

To remove a property use:

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

To get a property's value from a style:

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

`lv_style_value_t` has 3 fields, only one of which will apply, depending on the type of property it is applied to:

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

To reset a style (free all its data) use:

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

Styles can be built as `const` as well to save RAM:

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

LV_STYLE_CONST_INIT(style1, style1_props);
```

Later `const` style can be used like any other style but (obviously) new properties cannot be added.

Add and remove styles to a widget

A style on its own has no effect until it is added (assigned) to a Widget.

Add styles

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

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

Using *lv_obj_add_style()*:

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

Replace styles

To replace a specific style of a Widget use *lv_obj_replace_style*`(widget, old_style, new_style, selector)`. This function will only replace `old_style` with `new_style` if the `selector` matches the `selector` used in `lv_obj_add_style`. Both `old_style` and `new_style` must not be `NULL`. Separate functions exist for adding and removing styles. If the combination of `old_style` and `selector` exists multiple times in `obj`'s styles, all occurrences will be replaced. The return value of the function indicates whether at least one successful replacement took place.

Using *lv_obj_replace_style()*:

```
lv_obj_add_style(btn, &style_btn, 0);                     /* Add a button␣
↪style */
lv_obj_replace_style(btn, &style_btn, &new_style_btn, 0);  /* Replace the␣
↪button style with a different one */
```

Remove styles

To remove all styles from a Widget use *lv_obj_remove_style_all*`(widget)`.

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

Reporting style changes

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

1. If you know that the changed properties can be applied by a simple redraw (e.g. color or opacity changes) just call *lv_obj_invalidate*`(widget)` or *lv_obj_invalidate*(*lv_screen_active*()).
2. If more complex style properties were changed or added, and you know which Widget(s) are affected by that style call *lv_obj_refresh_style*`(widget, part, property)`. To refresh all parts and properties use *lv_obj_refresh_style*`(widget,` *LV_PART_ANY*`,` *LV_STYLE_PROP_ANY*`)`.

3. To make LVGL check all Widgets to see if they use a style and refresh them when needed, call `lv_obj_report_style_change(&style)`. If `style` is `NULL` all Widgets will be notified about a style change.

Get a style property's value on a Widget

To get the final value of a style's property considering

- cascading,
- inheritance,
- local styles and transitions (see below)

property "get" functions like this can be used: `lv_obj_get_style_<property_name>(widget, <part>)`. These functions use the Widget's current state and if no better candidate exists they return the default value. Ad esempio:

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

Local Styles

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

Local styles are like normal styles, but they can't be shared among other Widgets. If used, local styles are allocated automatically, and freed when the Widget is deleted. They are useful to add local customization to a Widget.

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

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

```
lv_obj_set_style_bg_color(slider, lv_color_red(), LV_PART_INDICATOR | LV_
↪STATE_FOCUSED);
```

Style Properties Overview

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

Typical background properties

In documentation of widgets you will see sentences like "The _____ Widget uses the typical background style properties". These "typical background properties" are the properties being referred to:

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

See *Boxing Model* for the meanings of these terms.

Transitions

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

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

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

The transition properties can be defined for each state. For example, setting a 500 ms transition time in the default state means that when the Widget goes to the default state a 500 ms transition time is applied. Setting a 100 ms transition time in the pressed state causes a 100 ms transition when going to the pressed state. This example configuration results in going to the pressed state quickly and then going back to default slowly.

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

```
/* Only its pointer is saved so must static, global or dynamically␣
↪allocated */
static const lv_style_prop_t trans_props[] = {
                                            LV_STYLE_BG_OPA, LV_STYLE_BG_
↪COLOR,
                                            0, /* End marker */
};

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

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

Opacity, Blend Modes and Transformations

If the `opa`, `blend_mode`, `transform_angle`, or `transform_zoom` properties are set to a non-default value LVGL creates a snapshot of the widget and its children in order to blend the whole widget with the set opacity, blend mode and transformation properties.

These properties have this effect only on the `MAIN` part of the widget.

The created snapshot is called "intermediate layer" or simply "layer". If only `opa` and/or `blend_mode` is set to a non-default value LVGL can build the layer from smaller chunks. The size of these chunks can be configured by the following properties in `lv_conf.h`:

- `LV_LAYER_SIMPLE_BUF_SIZE`: [bytes] the optimal target buffer size. LVGL will try to allocate this size of memory.
- `LV_LAYER_SIMPLE_FALLBACK_BUF_SIZE`: [bytes] used if `LV_LAYER_SIMPLE_BUF_SIZE` couldn't be allocated.

If transformation properties were also used the layer cannot be rendered in chunks, but one larger memory block needs to be allocated. The required memory depends on the angle, zoom and pivot parameters, and the size of the area to redraw, but it's never larger than the size of the widget (including the extra draw size used for shadow, outline, etc).

If the widget can fully cover the area to redraw, LVGL creates an RGB layer (which is faster to render and uses less memory). If the opposite case ARGB rendering needs to be used, a widget might not cover its area if it has radius, `bg_opa < 255`, has shadow, outline, etc.

The click area of the widget is also transformed accordingly.

Color Filter

TODO

Temi

Themes are a collection of styles. If there is an active theme LVGL applies it to every newly-created widget. This will give a default appearance to the UI which can then be modified by adding further styles.

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

To set a theme for a display, two steps are required:

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

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

```
lv_theme_t * th = lv_theme_default_init(display,                /* Use␣
↪DPI, size, etc. from this display */
                                        LV_COLOR_PALETTE_BLUE,  /*␣
↪Primary and secondary palette */
                                        LV_COLOR_PALETTE_CYAN,
                                        false,                  /* Dark␣
↪theme?  False = light theme. */
                                        &lv_font_montserrat_10, /* Small,␣
↪normal, large fonts */
                                        &lv_font_montserrat_14,
                                        &lv_font_montserrat_18);

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

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

Extending themes

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

`lv_theme_set_parent``(new_theme, base_theme)` extends the `base_theme` with the `new_theme`.

There is an example of this below.

Esempi

Stili Size

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_IMAGE

/**
 * Using the Size, Position and Padding style properties
 */
void lv_example_style_1(void)
{
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_radius(&style, 5);
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Make a gradient*/
    lv_style_set_width(&style, 150);
    lv_style_set_height(&style, LV_SIZE_CONTENT);

    lv_style_set_pad_ver(&style, 20);
    lv_style_set_pad_left(&style, 5);

    lv_style_set_x(&style, lv_pct(50));
    lv_style_set_y(&style, 80);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);

    lv_obj_t * label = lv_label_create(obj);
    lv_label_set_text(label, "Hello");
}

#endif
```

Stili Background

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

/**
 * Using the background style properties
 */
void lv_example_style_2(void)
{
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_radius(&style, 5);

    /*Make a gradient*/
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    static lv_grad_dsc_t grad;
    grad.dir = LV_GRAD_DIR_VER;
    grad.stops_count = 2;
    grad.stops[0].color = lv_palette_lighten(LV_PALETTE_GREY, 1);
    grad.stops[0].opa = LV_OPA_COVER;
    grad.stops[1].color = lv_palette_main(LV_PALETTE_BLUE);
    grad.stops[1].opa = LV_OPA_COVER;

    /*Shift the gradient to the bottom*/
    grad.stops[0].frac  = 128;
    grad.stops[1].frac  = 192;

    lv_style_set_bg_grad(&style, &grad);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_center(obj);
```

(continues on next page)

(continua dalla pagina precedente)

```
}

#endif
```

Stili Border

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

/**
 * Using the border style properties
 */
void lv_example_style_3(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    /*Set a background color and a radius*/
    lv_style_set_radius(&style, 10);
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_bg_color(&style, lv_palette_lighten(LV_PALETTE_GREY, 1));

    /*Add border to the bottom+right*/
    lv_style_set_border_color(&style, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_border_width(&style, 5);
    lv_style_set_border_opa(&style, LV_OPA_50);
    lv_style_set_border_side(&style, (lv_border_side_t)(LV_BORDER_SIDE_
↪BOTTOM | LV_BORDER_SIDE_RIGHT));

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_center(obj);
}

#endif
```

Stili Outline

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

/**
 * Using the outline style properties
 */
void lv_example_style_4(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    /*Set a background color and a radius*/
    lv_style_set_radius(&style, 5);
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_bg_color(&style, lv_palette_lighten(LV_PALETTE_GREY, 1));
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Add outline*/
    lv_style_set_outline_width(&style, 2);
    lv_style_set_outline_color(&style, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_outline_pad(&style, 8);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_center(obj);
}

#endif
```

Stili Shadow

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

/**
 * Using the Shadow style properties
 */
void lv_example_style_5(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    /*Set a background color and a radius*/
    lv_style_set_radius(&style, 5);
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_bg_color(&style, lv_palette_lighten(LV_PALETTE_GREY, 1));

    /*Add a shadow*/
    lv_style_set_shadow_width(&style, 55);
    lv_style_set_shadow_color(&style, lv_palette_main(LV_PALETTE_BLUE));

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_center(obj);
}

#endif
```

Stili Image

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_IMAGE

/**
 * Using the Image style properties
 */
void lv_example_style_6(void)
{
    static lv_style_t style;
    lv_style_init(&style);
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Set a background color and a radius*/
    lv_style_set_radius(&style, 5);
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_bg_color(&style, lv_palette_lighten(LV_PALETTE_GREY, 3));
    lv_style_set_border_width(&style, 2);
    lv_style_set_border_color(&style, lv_palette_main(LV_PALETTE_BLUE));

    lv_style_set_image_recolor(&style, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_image_recolor_opa(&style, LV_OPA_50);
    lv_style_set_transform_rotation(&style, 300);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_image_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);

    LV_IMAGE_DECLARE(img_cogwheel_argb);
    lv_image_set_src(obj, &img_cogwheel_argb);

    lv_obj_center(obj);
}

#endif
```

Stili Arc

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

/**
 * Using the Arc style properties
 */
void lv_example_style_7(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    lv_style_set_arc_color(&style, lv_palette_main(LV_PALETTE_RED));
    lv_style_set_arc_width(&style, 4);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_arc_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_center(obj);
}
#endif
```

Stili Text

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_LABEL

/**
 * Using the text style properties
 */
```

(continues on next page)

(continua dalla pagina precedente)

```
void lv_example_style_8(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    lv_style_set_radius(&style, 5);
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_bg_color(&style, lv_palette_lighten(LV_PALETTE_GREY, 2));
    lv_style_set_border_width(&style, 2);
    lv_style_set_border_color(&style, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_pad_all(&style, 10);

    lv_style_set_text_color(&style, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_text_letter_space(&style, 5);
    lv_style_set_text_line_space(&style, 20);
    lv_style_set_text_decor(&style, LV_TEXT_DECOR_UNDERLINE);

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

    lv_obj_center(obj);
}

#endif
```

Stili Line

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_LINE

/**
 * Using the line style properties
 */
void lv_example_style_9(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    lv_style_set_line_color(&style, lv_palette_main(LV_PALETTE_GREY));
    lv_style_set_line_width(&style, 6);
    lv_style_set_line_rounded(&style, true);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_line_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);

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

    lv_obj_center(obj);
}

#endif
```

Transizione

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_IMAGE

/**
 * Creating a transition
 */
void lv_example_style_10(void)
{
    static const lv_style_prop_t props[] = {LV_STYLE_BG_COLOR, LV_STYLE_
↪BORDER_COLOR, LV_STYLE_BORDER_WIDTH, 0};

    /* A default transition
     * Make it fast (100ms) and start with some delay (200 ms)*/
    static lv_style_transition_dsc_t trans_def;
    lv_style_transition_dsc_init(&trans_def, props, lv_anim_path_linear,␣
↪100, 200, NULL);

    /* A special transition when going to pressed state
     * Make it slow (500 ms) but start  without delay*/
    static lv_style_transition_dsc_t trans_pr;
    lv_style_transition_dsc_init(&trans_pr, props, lv_anim_path_linear,␣
↪500, 0, NULL);

    static lv_style_t style_def;
    lv_style_init(&style_def);
    lv_style_set_transition(&style_def, &trans_def);

    static lv_style_t style_pr;
    lv_style_init(&style_pr);
    lv_style_set_bg_color(&style_pr, lv_palette_main(LV_PALETTE_RED));
    lv_style_set_border_width(&style_pr, 6);
    lv_style_set_border_color(&style_pr, lv_palette_darken(LV_PALETTE_RED,␣
↪3));
    lv_style_set_transition(&style_pr, &trans_pr);

    /*Create an object with the new style_pr*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style_def, 0);
    lv_obj_add_style(obj, &style_pr, LV_STATE_PRESSED);

    lv_obj_center(obj);
}

#endif
```

Utilizzo di stili multipli

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_IMAGE

/**
 * Using multiple styles
 */
void lv_example_style_11(void)
```

(continues on next page)

(continua dalla pagina precedente)

```
{
    /*A base style*/
    static lv_style_t style_base;
    lv_style_init(&style_base);
    lv_style_set_bg_color(&style_base, lv_palette_main(LV_PALETTE_LIGHT_
↪BLUE));
    lv_style_set_border_color(&style_base, lv_palette_darken(LV_PALETTE_
↪LIGHT_BLUE, 3));
    lv_style_set_border_width(&style_base, 2);
    lv_style_set_radius(&style_base, 10);
    lv_style_set_shadow_width(&style_base, 10);
    lv_style_set_shadow_offset_y(&style_base, 5);
    lv_style_set_shadow_opa(&style_base, LV_OPA_50);
    lv_style_set_text_color(&style_base, lv_color_white());
    lv_style_set_width(&style_base, 100);
    lv_style_set_height(&style_base, LV_SIZE_CONTENT);

    /*Set only the properties that should be different*/
    static lv_style_t style_warning;
    lv_style_init(&style_warning);
    lv_style_set_bg_color(&style_warning, lv_palette_main(LV_PALETTE_
↪YELLOW));
    lv_style_set_border_color(&style_warning, lv_palette_darken(LV_PALETTE_
↪YELLOW, 3));
    lv_style_set_text_color(&style_warning, lv_palette_darken(LV_PALETTE_
↪YELLOW, 4));

    /*Create an object with the base style only*/
    lv_obj_t * obj_base = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj_base, &style_base, 0);
    lv_obj_align(obj_base, LV_ALIGN_LEFT_MID, 20, 0);

    lv_obj_t * label = lv_label_create(obj_base);
    lv_label_set_text(label, "Base");
    lv_obj_center(label);

    /*Create another object with the base style and earnings style too*/
    lv_obj_t * obj_warning = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj_warning, &style_base, 0);
    lv_obj_add_style(obj_warning, &style_warning, 0);
    lv_obj_align(obj_warning, LV_ALIGN_RIGHT_MID, -20, 0);

    label = lv_label_create(obj_warning);
    lv_label_set_text(label, "Warning");
    lv_obj_center(label);
}

#endif
```

Stili Locali

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_IMAGE

/**
```

(continues on next page)

(continua dalla pagina precedente)

```
 * Local styles
 */
void lv_example_style_12(void)
{
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_bg_color(&style, lv_palette_main(LV_PALETTE_GREEN));
    lv_style_set_border_color(&style, lv_palette_lighten(LV_PALETTE_GREEN,␣
↪3));
    lv_style_set_border_width(&style, 3);

    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);

    /*Overwrite the background color locally*/
    lv_obj_set_style_bg_color(obj, lv_palette_main(LV_PALETTE_ORANGE), LV_
↪PART_MAIN);

    lv_obj_center(obj);
}

#endif
```

Aggiunta di stili a parti e a stati

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_IMAGE

/**
 * Add styles to parts and states
 */
void lv_example_style_13(void)
{
    static lv_style_t style_indic;
    lv_style_init(&style_indic);
    lv_style_set_bg_color(&style_indic, lv_palette_lighten(LV_PALETTE_RED,␣
↪3));
    lv_style_set_bg_grad_color(&style_indic, lv_palette_main(LV_PALETTE_
↪RED));
    lv_style_set_bg_grad_dir(&style_indic, LV_GRAD_DIR_HOR);

    static lv_style_t style_indic_pr;
    lv_style_init(&style_indic_pr);
    lv_style_set_shadow_color(&style_indic_pr, lv_palette_main(LV_PALETTE_
↪RED));
    lv_style_set_shadow_width(&style_indic_pr, 10);
    lv_style_set_shadow_spread(&style_indic_pr, 3);

    /*Create an object with the new style_pr*/
    lv_obj_t * obj = lv_slider_create(lv_screen_active());
    lv_obj_add_style(obj, &style_indic, LV_PART_INDICATOR);
    lv_obj_add_style(obj, &style_indic_pr, LV_PART_INDICATOR | LV_STATE_
↪PRESSED);
    lv_slider_set_value(obj, 70, LV_ANIM_OFF);
    lv_obj_center(obj);
```

(continues on next page)

(continua dalla pagina precedente)

```
}

#endif
```

Estensione del tema corrente

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_IMAGE
#include "../../lvgl_private.h"

static lv_style_t style_btn;

/*Will be called when the styles of the base theme are already added
  to add new styles*/
static void new_theme_apply_cb(lv_theme_t * th, lv_obj_t * obj)
{
    LV_UNUSED(th);

    if(lv_obj_check_type(obj, &lv_button_class)) {
        lv_obj_add_style(obj, &style_btn, 0);
    }
}

static void new_theme_init_and_set(void)
{
    /*Initialize the styles*/
    lv_style_init(&style_btn);
    lv_style_set_bg_color(&style_btn, lv_palette_main(LV_PALETTE_GREEN));
    lv_style_set_border_color(&style_btn, lv_palette_darken(LV_PALETTE_
↪GREEN, 3));
    lv_style_set_border_width(&style_btn, 3);

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

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

    /*Assign the new theme to the current display*/
    lv_display_set_theme(NULL, &th_new);
}

/**
 * Extending the current theme
 */
void lv_example_style_14(void)
{
    lv_obj_t * btn;
    lv_obj_t * label;

    btn = lv_button_create(lv_screen_active());
    lv_obj_align(btn, LV_ALIGN_TOP_MID, 0, 20);
```

(continues on next page)

(continua dalla pagina precedente)

```
    label = lv_label_create(btn);
    lv_label_set_text(label, "Original theme");

    new_theme_init_and_set();

    btn = lv_button_create(lv_screen_active());
    lv_obj_align(btn, LV_ALIGN_BOTTOM_MID, 0, -20);

    label = lv_label_create(btn);
    lv_label_set_text(label, "New theme");
}

#endif
```

Opacità e trasformazioni

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_BUTTON && LV_USE_LABEL

/**
 * Opacity and Transformations
 */
void lv_example_style_15(void)
{
    lv_obj_t * btn;
    lv_obj_t * label;

    /*Normal button*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_set_size(btn, 100, 40);
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, -70);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Normal");
    lv_obj_center(label);

    /*Set opacity
     *The button and the label is rendered to a layer first and that layer␣
↪is blended*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_set_size(btn, 100, 40);
    lv_obj_set_style_opa(btn, LV_OPA_50, 0);
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, 0);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Opa:50%");
    lv_obj_center(label);

    /*Set transformations
     *The button and the label is rendered to a layer first and that layer␣
↪is transformed*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_set_size(btn, 100, 40);
    lv_obj_set_style_transform_rotation(btn, 150, 0);        /*15 deg*/
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_style_transform_scale(btn, 256 + 64, 0);   /*1.25x*/
    lv_obj_set_style_transform_pivot_x(btn, 50, 0);
    lv_obj_set_style_transform_pivot_y(btn, 20, 0);
    lv_obj_set_style_opa(btn, LV_OPA_50, 0);
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, 70);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Transf.");
    lv_obj_center(label);
}

#endif
```

Manopola metallica con gradiente conico

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

#if LV_USE_DRAW_SW_COMPLEX_GRADIENTS

/**
 * Simulate metallic knob using conical gradient
 * For best effect set LV_GRADIENT_MAX_STOPS to 8 or at least 3
 */
void lv_example_style_16(void)
{
#if LV_GRADIENT_MAX_STOPS >= 8
    static const lv_color_t grad_colors[8] = {
        LV_COLOR_MAKE(0xe8, 0xe8, 0xe8),
        LV_COLOR_MAKE(0xff, 0xff, 0xff),
        LV_COLOR_MAKE(0xfa, 0xfa, 0xfa),
        LV_COLOR_MAKE(0x79, 0x79, 0x79),
        LV_COLOR_MAKE(0x48, 0x48, 0x48),
        LV_COLOR_MAKE(0x4b, 0x4b, 0x4b),
        LV_COLOR_MAKE(0x70, 0x70, 0x70),
        LV_COLOR_MAKE(0xe8, 0xe8, 0xe8),
    };
#elif LV_GRADIENT_MAX_STOPS >= 3
    static const lv_color_t grad_colors[3] = {
        LV_COLOR_MAKE(0xe8, 0xe8, 0xe8),
        LV_COLOR_MAKE(0xff, 0xff, 0xff),
        LV_COLOR_MAKE(0x79, 0x79, 0x79),
    };
#else
    static const lv_color_t grad_colors[2] = {
        LV_COLOR_MAKE(0xe8, 0xe8, 0xe8),
        LV_COLOR_MAKE(0x79, 0x79, 0x79),
    };
#endif

    /*Create a style with gradient background and shadow*/
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_radius(&style, 500);
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_set_shadow_color(&style, lv_color_black());
    lv_style_set_shadow_width(&style, 50);
    lv_style_set_shadow_offset_x(&style, 20);
    lv_style_set_shadow_offset_y(&style, 20);
    lv_style_set_shadow_opa(&style, LV_OPA_50);

    /*First define a color gradient. In this example we use a gray color␣
↪map with random values.*/
    static lv_grad_dsc_t grad;

    lv_grad_init_stops(&grad, grad_colors, NULL, NULL, sizeof(grad_colors)␣
↪/ sizeof(lv_color_t));

    /*Make a conical gradient with the center in the middle of the object*/
#if LV_GRADIENT_MAX_STOPS >= 8
    lv_grad_conical_init(&grad, LV_GRAD_CENTER, LV_GRAD_CENTER, 0, 120, LV_
↪GRAD_EXTEND_REFLECT);
#elif LV_GRADIENT_MAX_STOPS >= 3
    lv_grad_conical_init(&grad, LV_GRAD_CENTER, LV_GRAD_CENTER, 45, 125,␣
↪LV_GRAD_EXTEND_REFLECT);
#else
    lv_grad_conical_init(&grad, LV_GRAD_CENTER, LV_GRAD_CENTER, 45, 110,␣
↪LV_GRAD_EXTEND_REFLECT);
#endif

    /*Set gradient as background*/
    lv_style_set_bg_grad(&style, &grad);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_set_size(obj, 200, 200);
    lv_obj_center(obj);
}

#else

void lv_example_style_16(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_set_width(label, LV_PCT(80));
    lv_label_set_text(label, "LV_USE_DRAW_SW_COMPLEX_GRADIENTS is not␣
↪enabled");
    lv_label_set_long_mode(label, LV_LABEL_LONG_MODE_SCROLL_CIRCULAR);
    lv_obj_center(label);
}

#endif /*LV_USE_DRAW_SW_COMPLEX_GRADIENTS*/

#endif /*LV_BUILD_EXAMPLES*/
```

Gradiente radiale come sfondo

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

(continues on next page)

(continua dalla pagina precedente)

```
#if LV_USE_DRAW_SW_COMPLEX_GRADIENTS

/**
 * Using radial gradient as background
 */
void lv_example_style_17(void)
{
    static const lv_color_t grad_colors[2] = {
        LV_COLOR_MAKE(0x9B, 0x18, 0x42),
        LV_COLOR_MAKE(0x00, 0x00, 0x00),
    };

    int32_t width = lv_display_get_horizontal_resolution(NULL);
    int32_t height = lv_display_get_vertical_resolution(NULL);

    static lv_style_t style;
    lv_style_init(&style);

    /*First define a color gradient. In this example we use a purple to␣
↪black color map.*/
    static lv_grad_dsc_t grad;

    lv_grad_init_stops(&grad, grad_colors, NULL, NULL, sizeof(grad_colors)␣
↪/ sizeof(lv_color_t));

    /*Make a radial gradient with the center in the middle of the object,␣
↪extending to the farthest corner*/
    lv_grad_radial_init(&grad, LV_GRAD_CENTER, LV_GRAD_CENTER, LV_GRAD_
↪RIGHT, LV_GRAD_BOTTOM, LV_GRAD_EXTEND_PAD);

    /*Set gradient as background*/
    lv_style_set_bg_grad(&style, &grad);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_set_size(obj, width, height);
    lv_obj_center(obj);
}

#else

void lv_example_style_17(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_set_width(label, LV_PCT(80));
    lv_label_set_text(label, "LV_USE_DRAW_SW_COMPLEX_GRADIENTS is not␣
↪enabled");
    lv_label_set_long_mode(label, LV_LABEL_LONG_MODE_SCROLL_CIRCULAR);
    lv_obj_center(label);
}

#endif /*LV_USE_DRAW_SW_COMPLEX_GRADIENTS*/

#endif /*LV_BUILD_EXAMPLES*/
```

Gradienti per lo sfondo del pulsante

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

#if LV_USE_DRAW_SW_COMPLEX_GRADIENTS

/**
 * Using various gradients for button background
 */
void lv_example_style_18(void)
{
    static const lv_color_t grad_colors[2] = {
        LV_COLOR_MAKE(0x26, 0xa0, 0xda),
        LV_COLOR_MAKE(0x31, 0x47, 0x55),
    };

    /*Create a linear gradient going from the top left corner to the␣
↪bottom at an angle, with reflected color map*/
    static lv_style_t style_with_linear_gradient_bg;
    static lv_grad_dsc_t linear_gradient_dsc;     /*NOTE: the gradient␣
↪descriptor must be static or global variable!*/

    lv_style_init(&style_with_linear_gradient_bg);
    lv_grad_init_stops(&linear_gradient_dsc, grad_colors, NULL, NULL,␣
↪sizeof(grad_colors) / sizeof(lv_color_t));
    lv_grad_linear_init(&linear_gradient_dsc, lv_pct(0), lv_pct(0), lv_
↪pct(20), lv_pct(100), LV_GRAD_EXTEND_REFLECT);
    lv_style_set_bg_grad(&style_with_linear_gradient_bg, &linear_gradient_
↪dsc);
    lv_style_set_bg_opa(&style_with_linear_gradient_bg, LV_OPA_COVER);

    /*Create a radial gradient with the center in the top left 1/3rd of␣
↪the object, extending to the bottom right corner, with reflected color␣
↪map*/
    static lv_style_t style_with_radial_gradient_bg;
    static lv_grad_dsc_t radial_gradient_dsc;     /*NOTE: the gradient␣
↪descriptor must be static or global variable!*/

    lv_style_init(&style_with_radial_gradient_bg);
    lv_grad_init_stops(&radial_gradient_dsc, grad_colors, NULL, NULL,␣
↪sizeof(grad_colors) / sizeof(lv_color_t));
    lv_grad_radial_init(&radial_gradient_dsc, lv_pct(30), lv_pct(30), lv_
↪pct(100), lv_pct(100), LV_GRAD_EXTEND_REFLECT);
    lv_style_set_bg_grad(&style_with_radial_gradient_bg, &radial_gradient_
↪dsc);
    lv_style_set_bg_opa(&style_with_radial_gradient_bg, LV_OPA_COVER);

    /*Create buttons with different gradient styles*/

    lv_obj_t * btn;
    lv_obj_t * label;

    /*Simple horizontal gradient*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_set_style_bg_color(btn, grad_colors[0], 0);
    lv_obj_set_style_bg_grad_color(btn, grad_colors[1], 0);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_style_bg_grad_dir(btn, LV_GRAD_DIR_HOR, 0);
    lv_obj_set_size(btn, 150, 50);
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, -100);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Horizontal");
    lv_obj_center(label);

    /*Simple vertical gradient*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_set_style_bg_color(btn, grad_colors[0], 0);
    lv_obj_set_style_bg_grad_color(btn, grad_colors[1], 0);
    lv_obj_set_style_bg_grad_dir(btn, LV_GRAD_DIR_VER, 0);
    lv_obj_set_size(btn, 150, 50);
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, -40);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Vertical");
    lv_obj_center(label);

    /*Complex linear gradient*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_add_style(btn, &style_with_linear_gradient_bg, 0);
    lv_obj_set_size(btn, 150, 50);
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, 20);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Linear");
    lv_obj_center(label);

    /*Complex radial gradient*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_add_style(btn, &style_with_radial_gradient_bg, 0);
    lv_obj_set_size(btn, 150, 50);
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, 80);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Radial");
    lv_obj_center(label);
}

#else

void lv_example_style_18(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_set_width(label, LV_PCT(80));
    lv_label_set_text(label, "LV_USE_DRAW_SW_COMPLEX_GRADIENTS is not␣
↪enabled");
    lv_label_set_long_mode(label, LV_LABEL_LONG_MODE_SCROLL_CIRCULAR);
    lv_obj_center(label);
}

#endif /*LV_USE_DRAW_SW_COMPLEX_GRADIENTS*/

#endif /*LV_BUILD_EXAMPLES*/
```

Test tra stile di ricolorazione o sfondo modale pieno

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_SLIDER && LV_USE_LOG

/**
 * Test between a full background modal and a recolor modal
 */
void lv_example_style_19(void)
{
    /*Add lv_example_style_11 as background*/
    lv_example_style_11();

    /* Set to 1 to enable recolor overlay instead of solid background */
#if 0
    /* Apply a screen-wide tint using recolor (efficient overlay).
     * This modifies the visual appearance by blending a semi-transparent␣
↪color
     * over existing content without creating additional objects.
     * It's lighter on performance compared to a full-size background␣
↪object. */
    lv_obj_set_style_recolor(lv_screen_active(), lv_color_black(), 0);
    lv_obj_set_style_recolor_opa(lv_screen_active(), LV_OPA_50, 0);
#else
    /* Simulate a modal background by setting a semi-transparent black␣
↪background
     * on lv_layer_top(), the highest built-in layer.
     * This method creates a new full-screen object and can consume more␣
↪resources
     * compared to recolor, especially when using images or gradients. */
    lv_obj_set_style_bg_color(lv_layer_top(), lv_color_black(), 0);
    lv_obj_set_style_bg_opa(lv_layer_top(), LV_OPA_50, 0);
#endif

    lv_obj_t * obj = lv_slider_create(lv_layer_top());
    lv_obj_center(obj);

    lv_refr_now(NULL); /*Update layouts and render*/

    lv_obj_invalidate(lv_screen_active());

    uint32_t t = lv_tick_get();
    lv_refr_now(NULL); /*Render only*/
    LV_LOG_USER("%" LV_PRIu32 " ms\n", lv_tick_elaps(t));
}

#endif
```

API

lv_api_map_v8.h

lv_flex.h

lv_grid.h

lv_obj.h

lv_obj_pos.h

lv_obj_scroll.h

lv_obj_style.h

lv_obj_style_gen.h

lv_obj_tree.h

lv_style.h

lv_style_gen.h

5.4.2 Style Properties

Size and position

Properties related to size, position, alignment and layout of Widgets.

width

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

min_width

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

max_width

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

height

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

min_height

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

max_height

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

length

Its meaning depends on the type of Widget. For example in case of lv_scale it means the length of the ticks.

x

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

y

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

align

Set the alignment which tells from which point of the parent the X and Y coordinates should be interpreted. Possible values are: `LV_ALIGN_DEFAULT`, `LV_ALIGN_TOP_LEFT/MID/RIGHT`, `LV_ALIGN_BOTTOM_LEFT/MID/RIGHT`, `LV_ALIGN_LEFT/RIGHT_MID`, `LV_ALIGN_CENTER`. `LV_ALIGN_DEFAULT` means `LV_ALIGN_TOP_LEFT` with LTR base direction and `LV_ALIGN_TOP_RIGHT` with RTL base direction.

transform_width

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

transform_height

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

translate_x

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

translate_y

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

translate_radial

Move object around the centre of the parent object (e.g. around the circumference of a scale)

transform_scale_x

Zoom Widget horizontally. The value 256 (or `LV_SCALE_NONE`) means normal size, 128 half size, 512 double size, and so on

transform_scale_y

Zoom Widget vertically. The value 256 (or `LV_SCALE_NONE`) means normal size, 128 half size, 512 double size, and so on

transform_rotation

Rotate Widget. The value is interpreted in 0.1 degree units. E.g. 450 means 45 deg.

transform_pivot_x

Set pivot point's X coordinate for transformations. Relative to Widget's top left corner'

transform_pivot_y

Set pivot point's Y coordinate for transformations. Relative to Widget's top left corner'

transform_skew_x

Skew Widget horizontally. The value is interpreted in 0.1 degree units. E.g. 450 means 45 deg.

transform_skew_y

Skew Widget vertically. The value is interpreted in 0.1 degree units. E.g. 450 means 45 deg.

Padding

Properties to describe spacing between the parent's sides and the children and among the children. Very similar to the padding properties in HTML.

pad_top

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

pad_bottom

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

pad_left

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

pad_right

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

pad_row

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

pad_column

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

pad_radial

Pad text labels away from the scale ticks/remainder of the `LV_PART_`

Margin

Properties to describe spacing around a Widget. Very similar to the margin properties in HTML.

margin_top

Sets margin on the top. Widget will keep this space from its siblings in layouts.

margin_bottom

Sets margin on the bottom. Widget will keep this space from its siblings in layouts.

margin_left

Sets margin on the left. Widget will keep this space from its siblings in layouts.

margin_right

Sets margin on the right. Widget will keep this space from its siblings in layouts.

Background

Properties to describe the background color and image of Widget.

bg_color

Set background color of Widget.

bg_opa

Set opacity of the background. Value 0, `LV_OPA_0` or `LV_OPA_TRANSP` means fully transparent, 255, `LV_OPA_100` or `LV_OPA_COVER` means fully covering, other values or LV_OPA_10, LV_OPA_20, etc means semi transparency.

bg_grad_color

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

bg_grad_dir

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

bg_main_stop

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

bg_grad_stop

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

bg_main_opa

Set opacity of the first gradient color

bg_grad_opa

Set opacity of the second gradient color

bg_grad

Set gradient definition. The pointed instance must exist while Widget is alive. NULL to disable. It wraps `BG_GRAD_COLOR`, `BG_GRAD_DIR`, `BG_MAIN_STOP` and `BG_GRAD_STOP` into one descriptor and allows creating gradients with more colors as well. If it's set other gradient related properties will be ignored'

bg_image_src

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

bg_image_opa

Set opacity of the background image. Value 0, `LV_OPA_0` or `LV_OPA_TRANSP` means fully transparent, 255, `LV_OPA_100` or `LV_OPA_COVER` means fully covering, other values or LV_OPA_10, LV_OPA_20, etc means semi transparency.

bg_image_recolor

Set a color to mix to the background image.

bg_image_recolor_opa

Set intensity of background image recoloring. Value 0, `LV_OPA_0` or `LV_OPA_TRANSP` means no mixing, 255, `LV_OPA_100` or `LV_OPA_COVER` means full recoloring, other values or LV_OPA_10, LV_OPA_20, etc are interpreted proportionally.

bg_image_tiled

If enabled the background image will be tiled. Possible values are `true` or `false`.

Border

Properties to describe the borders

border_color

Set color of the border

border_opa

Set opacity of the border. Value 0, `LV_OPA_0` or `LV_OPA_TRANSP` means fully transparent, 255, `LV_OPA_100` or `LV_OPA_COVER` means fully covering, other values or LV_OPA_10, LV_OPA_20, etc means semi transparency.

border_width

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

border_side

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

border_post

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

Outline

Properties to describe the outline. It's like a border but drawn outside of the rectangles.

outline_width

Set width of outline in pixels.

outline_color

Set color of outline.

outline_opa

Set opacity of outline. Value 0, `LV_OPA_0` or `LV_OPA_TRANSP` means fully transparent, 255, `LV_OPA_100` or `LV_OPA_COVER` means fully covering, other values or LV_OPA_10, LV_OPA_20, etc means semi transparency.

outline_pad

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

Shadow

Properties to describe the shadow drawn under the rectangles.

shadow_width

Set width of the shadow in pixels. The value should be >= 0.

shadow_offset_x

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

shadow_offset_y

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

shadow_spread

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

shadow_color

Set color of shadow

shadow_opa

Set opacity of shadow. Value 0, `LV_OPA_0` or `LV_OPA_TRANSP` means fully transparent, 255, `LV_OPA_100` or `LV_OPA_COVER` means fully covering, other values or LV_OPA_10, LV_OPA_20, etc means semi transparency.

Immagine

Properties to describe the images

image_opa

Set opacity of an image. Value 0, `LV_OPA_0` or `LV_OPA_TRANSP` means fully transparent, 255, `LV_OPA_100` or `LV_OPA_COVER` means fully covering, other values or LV_OPA_10, LV_OPA_20, etc means semi transparency.

image_recolor

Set color to mix with the image.

image_recolor_opa

Set intensity of color mixing. Value 0, `LV_OPA_0` or `LV_OPA_TRANSP` means fully transparent, 255, `LV_OPA_100` or `LV_OPA_COVER` means fully covering, other values or LV_OPA_10, LV_OPA_20, etc means semi transparency.

Linea

Properties to describe line-like Widgets

line_width

Set width of lines in pixels.

line_dash_width

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

line_dash_gap

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

line_rounded

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

line_color

Set color of lines.

line_opa

Set opacity of lines.

Arco

TODO

arc_width

Set width (thickness) of arcs in pixels.

arc_rounded

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

arc_color

Set color of arc.

arc_opa

Set opacity of arcs.

arc_image_src

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

Text

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

text_color

Sets color of text.

text_opa

Set opacity of text. Value 0, `LV_OPA_0` or `LV_OPA_TRANSP` means fully transparent, 255, `LV_OPA_100` or `LV_OPA_COVER` means fully covering, other values or LV_OPA_10, LV_OPA_20, etc means semi transparency.

text_font

Set font of text (a pointer `lv_font_t *`).

text_letter_space

Set letter space in pixels

text_line_space

Set line space in pixels.

text_decor

Set decoration for the text. Possible values are `LV_TEXT_DECOR_NONE/UNDERLINE/STRIKETHROUGH`. OR-ed values can be used as well.

text_align

Set how to align the lines of the text. Note that it doesn't align the Widget itself, only the lines inside the Widget. Possible values are `LV_TEXT_ALIGN_LEFT/CENTER/RIGHT/AUTO`. `LV_TEXT_ALIGN_AUTO` detect the text base direction and uses left or right alignment accordingly

text_outline_stroke_color

Sets the color of letter outline stroke.

text_outline_stroke_width

Set the letter outline stroke width in pixels.

text_outline_stroke_opa

Set the opacity of the letter outline stroke. Value 0, `LV_OPA_0` or `LV_OPA_TRANSP` means fully transparent, 255, `LV_OPA_100` or `LV_OPA_COVER` means fully covering, other values or LV_OPA_10, LV_OPA_20, etc means semi transparency.

Miscellaneous

Mixed properties for various purposes.

radius

Set radius on every corner. The value is interpreted in pixels (>= 0) or `LV_RADIUS_CIRCLE` for max. radius

radial_offset

Move start point of object (e.g. scale tick) radially

clip_corner

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

opa

Scale down all opacity values of the Widget by this factor. Value 0, `LV_OPA_0` or `LV_OPA_TRANSP` means fully transparent, 255, `LV_OPA_100` or `LV_OPA_COVER` means fully covering, other values or LV_OPA_10, LV_OPA_20, etc means semi transparency.

opa_layered

First draw Widget on the layer, then scale down layer opacity factor. Value 0, `LV_OPA_0` or `LV_OPA_TRANSP` means fully transparent, 255, `LV_OPA_100` or `LV_OPA_COVER` means fully covering, other values or LV_OPA_10, LV_OPA_20, etc means semi transparency.

color_filter_dsc

Mix a color with all colors of the Widget.

color_filter_opa

The intensity of mixing of color filter.

anim

Animation template for Widget's animation. Should be a pointer to `lv_anim_t`. The animation parameters are widget specific, e.g. animation time could be the E.g. blink time of the cursor on the Text Area or scroll time of a roller. See Widgets' documentation to learn more.

anim_duration

Animation duration in milliseconds. Its meaning is widget specific. E.g. blink time of the cursor on the Text Area or scroll time of a roller. See Widgets' documentation to learn more.

transition

An initialized `lv_style_transition_dsc_t` to describe a transition.

blend_mode

Describes how to blend the colors to the background. Possible values are `LV_BLEND_MODE_NORMAL/ADDITIVE/SUBTRACTIVE/MULTIPLY/DIFFERENCE`

layout

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

base_dir

Set base direction of Widget. Possible values are `LV_BIDI_DIR_LTR/RTL/AUTO`.

bitmap_mask_src

If set, a layer will be created for the widget and the layer will be masked with this A8 bitmap mask.

rotary_sensitivity

Adjust sensitivity for rotary encoders in 1/256 unit. It means, 128: slow down the rotary to half, 512: speeds up to double, 256: no change

Flex

Flex layout properties.

flex_flow

Defines in which direct the flex layout should arrange the children

flex_main_place

Defines how to align the children in the direction of flex flow

flex_cross_place

Defines how to align the children perpendicular to the direction of flex flow

flex_track_place

Defines how to align the tracks of the flow

flex_grow

Defines how much space to take proportionally from the free space of the Widget's track

Grid

Grid layout properties.

grid_column_dsc_array

An array to describe the columns of the grid. Should be LV_GRID_TEMPLATE_LAST terminated

grid_column_align

Defines how to distribute the columns

grid_row_dsc_array

An array to describe the rows of the grid. Should be LV_GRID_TEMPLATE_LAST terminated

grid_row_align

Defines how to distribute the rows.

grid_cell_column_pos

Set column in which Widget should be placed.

grid_cell_x_align

Set how to align Widget horizontally.

grid_cell_column_span

Set how many columns Widget should span. Needs to be >= 1.

grid_cell_row_pos

Set row in which Widget should be placed.

grid_cell_y_align

Set how to align Widget vertically.

grid_cell_row_span

Set how many rows Widget should span. Needs to be >= 1.

5.5 Eventi

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

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

Besides Widgets, events can registered from displays and input devices as well. It is not detailed below, but you can do this by changing the prefix of the functions from `lv_obj_` to `lv_display_` or `lv_indev_`.

5.5.1 Adding Events to a Widget

The user can assign callback functions to a widget to process events. In practice, it looks like this:

```
lv_obj_t * btn = lv_button_create(lv_screen_active());
lv_obj_add_event_cb(btn, my_event_cb, LV_EVENT_CLICKED, user_data);   /*␣
↪Assign an event callback */

...

static void my_event_cb(lv_event_t * event)
{
```

(continues on next page)

(continua dalla pagina precedente)

```
    printf("Clicked\n");
}
```

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

The last parameter of *lv_obj_add_event_cb()* is a pointer to any custom data that will be available in the event. NULL may be passed for this argument if there is no need to use that data when the event is processed. You can retrieve the pointer passed when setting the callback function like this:

```
my_user_data_t  * user_data;
...
user_data = lv_event_get_user_data(e);
```

More events can be added to a Widget, like this:

```
lv_obj_add_event_cb(widget, my_event_cb_1, LV_EVENT_CLICKED, NULL);
lv_obj_add_event_cb(widget, my_event_cb_2, LV_EVENT_PRESSED, NULL);
lv_obj_add_event_cb(widget, my_event_cb_3, LV_EVENT_ALL, NULL);       /*␣
↪No filtering, receive all events */
```

Even the same event callback can be used on a Widget with different `user_data`. Ad esempio:

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

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

Other Widgets can use the same *event callback*.

In the very same way, events can be attached to input devices and displays like this:

```
lv_display_add_event_cb(disp, event_cb, LV_EVENT_RESOLUTION_CHANGED, NULL);
lv_indev_add_event_cb(indev, event_cb, LV_EVENT_CLICKED, NULL);
```

5.5.2 Removing Event(s) from Widgets

```
uint32_t i;
uint32_t event_cnt = lv_obj_get_event_count(widget);
for(i = 0; i < event_cnt; i++) {
    lv_event_dsc_t * event_dsc = lv_obj_get_event_dsc(widget, i);
    if(lv_event_dsc_get_cb(event_dsc) == some_event_cb) {
        lv_obj_remove_event(widget, i);
        break;
    }
}
```

5.5.3 Event Codes

The event codes can be grouped into these categories: - Input device events - Drawing events - Other events - Special events - Custom events

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

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

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

The following event codes exist:

Input Device Events

- *LV_EVENT_PRESSED*: Widget has been pressed
- *LV_EVENT_PRESSING*: Widget is being pressed (called continuously while pressing)
- *LV_EVENT_PRESS_LOST*: Widget is still being pressed but slid cursor/finger off Widget
- *LV_EVENT_SHORT_CLICKED*: Widget was pressed for a short period of time, and then released without scrolling.
- *LV_EVENT_SINGLE_CLICKED*: Widget was pressed for a short period of time, and then released without scrolling, for the first time in a click streak. A click streak refers to multiple short clicks within a short period of time and a small distance.
- *LV_EVENT_DOUBLE_CLICKED*: Widget was pressed for a short period of time, and then released without scrolling, for the second time in a click streak.
- *LV_EVENT_TRIPLE_CLICKED*: Widget was pressed for a short period of time, and then released without scrolling, for the third time in a click streak.
- *LV_EVENT_LONG_PRESSED*: Widget has been pressed for at least `long_press_time`. Not called if scrolled.
- *LV_EVENT_LONG_PRESSED_REPEAT*: Called after `long_press_time` in every `long_press_repeat_time` ms. Not called if scrolled.
- *LV_EVENT_CLICKED*: Called on release if not scrolled (regardless of long press)
- *LV_EVENT_RELEASED*: Called in every cases when Widget has been released
- *LV_EVENT_SCROLL_BEGIN*: Scrolling begins. The event parameter is a pointer to the animation of the scroll. Can be modified
- *LV_EVENT_SCROLL_THROW_BEGIN*:
- *LV_EVENT_SCROLL_END*: Scrolling ends
- *LV_EVENT_SCROLL*: Scrolling
- *LV_EVENT_GESTURE*: A gesture is detected. Get the gesture with *lv_indev_get_gesture_dir*(*lv_indev_active*())
- *LV_EVENT_KEY*: A key is sent to Widget. Get the key with *lv_indev_get_key*(*lv_indev_active*())
- *LV_EVENT_FOCUSED*: Widget received focus
- *LV_EVENT_DEFOCUSED*: Widget is defocused
- *LV_EVENT_LEAVE*: Widget is defocused but still selected
- *LV_EVENT_HIT_TEST*: Perform advanced hit-testing
- *LV_EVENT_INDEV_RESET*: Indev has been reset
- *LV_EVENT_HOVER_OVER*: Indev hover over Widget
- *LV_EVENT_HOVER_LEAVE*: Indev hover leave Widget

Drawing Events

- *LV_EVENT_COVER_CHECK*: Check if Widget fully covers an area. The event parameter is `lv_cover_check_info_t *`.
- *LV_EVENT_REFR_EXT_DRAW_SIZE*: Get the required extra draw area around Widget (e.g. for shadow). The event parameter is `int32_t *` to store the size.
- *LV_EVENT_DRAW_MAIN_BEGIN*: Starting the main drawing phase

- *LV_EVENT_DRAW_MAIN*: Perform the main drawing
- *LV_EVENT_DRAW_MAIN_END*: Finishing the main drawing phase
- *LV_EVENT_DRAW_POST_BEGIN*: Starting the post draw phase (when all children are drawn)
- *LV_EVENT_DRAW_POST*: Perform the post draw phase (when all children are drawn)
- *LV_EVENT_DRAW_POST_END*: Finishing the post draw phase (when all children are drawn)
- *LV_EVENT_DRAW_TASK_ADDED*: Adding a draw task

> **Attenzione**
>
> In drawing-event callback functions the rendering sequence has already begun, and during this period, making changes to any Widget's attributes, styles, or creating/deleting widgets is forbidden. If you attempt to do so, LVGL will generate an assertion failure with a message indicating that invalidating an area is not allowed during rendering.

Special Events

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

Other Events

- *LV_EVENT_CREATE*: Widget is being created
- *LV_EVENT_DELETE*: Widget is being deleted
- *LV_EVENT_CHILD_CHANGED*: Child was removed, added, or its size, position were changed
- *LV_EVENT_CHILD_CREATED*: Child was created, always bubbles up to all parents
- *LV_EVENT_CHILD_DELETED*: Child was deleted, always bubbles up to all parents
- *LV_EVENT_SCREEN_UNLOAD_START*: A screen unload started, fired immediately when scr_load is called
- *LV_EVENT_SCREEN_LOAD_START*: A screen load started, fired when the screen change delay is expired
- *LV_EVENT_SCREEN_LOADED*: A screen was loaded
- *LV_EVENT_SCREEN_UNLOADED*: A screen was unloaded
- *LV_EVENT_SIZE_CHANGED*: Widget coordinates/size have changed
- *LV_EVENT_STYLE_CHANGED*: Widget's style has changed
- *LV_EVENT_LAYOUT_CHANGED*: The children position has changed due to a layout recalculation
- *LV_EVENT_GET_SELF_SIZE*: Get the internal size of a widget

Display Events

- *LV_EVENT_INVALIDATE_AREA* An area is invalidated (marked for redraw). *lv_event_get_param*`(e)` returns a pointer to an *lv_area_t* object with the coordinates of the area to be invalidated. The area can be freely modified if needed to adapt it a special requirement of the display. Usually needed with monochrome displays to invalidate `N x 8` rows or columns in one pass.
- *LV_EVENT_RESOLUTION_CHANGED*: Sent when the resolution changes due to *lv_display_set_resolution()* or *lv_display_set_rotation()*.

- *LV_EVENT_COLOR_FORMAT_CHANGED*: Sent as a result of any call to `lv_display_set_color_format()`.
- *LV_EVENT_REFR_REQUEST*: Sent when something happened that requires redraw.
- *LV_EVENT_REFR_START*: Sent before a refreshing cycle starts. Sent even if there is nothing to redraw.
- *LV_EVENT_REFR_READY*: Sent when refreshing has been completed (after rendering and calling *Flush Callback*). Sent even if no redraw happened.
- *LV_EVENT_RENDER_START*: Sent just before rendering begins.
- *LV_EVENT_RENDER_READY*: Sent after rendering has been completed (before calling *Flush Callback*)
- *LV_EVENT_FLUSH_START*: Sent before *Flush Callback* is called.
- *LV_EVENT_FLUSH_FINISH*: Sent after *Flush Callback* call has returned.
- *LV_EVENT_FLUSH_WAIT_START*: Sent at the beginning of internal call to `wait_for_flushing()` -- happens whether or not any waiting will actually occur. Call returns immediately if `disp->flushing == 0`.
- *LV_EVENT_FLUSH_WAIT_FINISH*: Sent when the call to `wait_for_flushing()` is about to return, regardless whether any actual waiting occurred.

Custom Events

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

They can be sent to any Widget with `lv_obj_send_event(widget, MY_EVENT_1, &some_data)`

Refresh Event

LV_EVENT_REFRESH is a special event because it's designed to let the user notify a Widget to refresh itself. Alcuni esempi:

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

5.5.4 Sending Events Manually

To manually send events to a Widget, use `lv_obj_send_event(widget, <EVENT_CODE>, &some_data)`.

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

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

The same works for display and input devices with `lv_display_send_event(widget, <EVENT_CODE>, &some_data)` and `lv_indev_send_event(widget, <EVENT_CODE>, &some_data)`.

5.5.5 Fields of lv_event_t

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

- *lv_event_get_code*`(e)`: get the event code

- `lv_event_get_current_target(e)`: get Widget to which an event was sent. I.e. the Widget whose event handler is being called.
- `lv_event_get_target(e)`: get Widget that originally triggered the event (different from `lv_event_get_target()` if *event bubbling* is enabled)
- `lv_event_get_user_data(e)`: get the pointer passed as the last parameter of `lv_obj_add_event_cb()`.
- `lv_event_get_param(e)`: get the parameter passed as the last parameter of `lv_obj_send_event_cb()`

5.5.6 Event Bubbling

If `lv_obj_add_flag(widget, LV_OBJ_FLAG_EVENT_BUBBLE)` is enabled all events will be sent to a Widget's parent as well. If the parent also has `LV_OBJ_FLAG_EVENT_BUBBLE` enabled the event will be sent to its parent, and so on.

The *target* parameter of the event is always the current target Widget, not the original Widget. To get the original target call `lv_event_get_target_obj(e)` in the event handler.

5.5.7 Event Trickle

Also known as Event Capturing, if `lv_obj_add_flag(widget, LV_OBJ_FLAG_EVENT_TRICKLE)` is enabled all events will be sent to the Widget's children as well. This is the opposite of event bubbling --- instead of propagating up the parent, events propagate down to the children.

The trickle mechanism only affects immediate children, not grandchildren or deeper descendants. If you need events to propagate to deeper levels, each child would need to have the `LV_OBJ_FLAG_EVENT_TRICKLE` flag enabled.

Like with bubbling, the *target* parameter of the event is always the current target Widget, not the original Widget. To get the original target call `lv_event_get_target_obj(e)` in the event handler.

5.5.8 Esempi

Evento clic pulsante

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_SWITCH

static void event_cb(lv_event_t * e)
{
    LV_LOG_USER("Clicked");

    static uint32_t cnt = 1;
    lv_obj_t * btn = lv_event_get_target_obj(e);
    lv_obj_t * label = lv_obj_get_child(btn, 0);
    lv_label_set_text_fmt(label, "%" LV_PRIu32, cnt);
    cnt++;
}

/**
 * Add click event to a button
 */
void lv_example_event_click(void)
{
    lv_obj_t * btn = lv_button_create(lv_screen_active());
    lv_obj_set_size(btn, 100, 50);
    lv_obj_center(btn);
    lv_obj_add_event_cb(btn, event_cb, LV_EVENT_CLICKED, NULL);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_t * label = lv_label_create(btn);
    lv_label_set_text(label, "Click me!");
    lv_obj_center(label);
}

#endif
```

Sequenze di clic

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_LABEL

static void short_click_event_cb(lv_event_t * e)
{
    LV_LOG_USER("Short clicked");

    lv_obj_t * info_label = (lv_obj_t *) lv_event_get_user_data(e);
    lv_indev_t * indev = (lv_indev_t *) lv_event_get_param(e);
    uint8_t cnt = lv_indev_get_short_click_streak(indev);
    lv_label_set_text_fmt(info_label, "Short click streak: %u", cnt);
}

static void streak_event_cb(lv_event_t * e)
{
    lv_obj_t * btn = lv_event_get_target_obj(e);
    lv_obj_t * label = lv_obj_get_child(btn, 0);
    const char * text = (const char *) lv_event_get_user_data(e);
    lv_label_set_text(label, text);
}

void lv_example_event_streak(void)
{
    lv_obj_t * info_label = lv_label_create(lv_screen_active());
    lv_label_set_text(info_label, "No events yet");

    lv_obj_t * btn = lv_button_create(lv_screen_active());
    lv_obj_set_size(btn, 100, 50);
    lv_obj_center(btn);
    lv_obj_add_event_cb(btn, short_click_event_cb, LV_EVENT_SHORT_CLICKED,␣
↪info_label);
    lv_obj_add_event_cb(btn, streak_event_cb, LV_EVENT_SINGLE_CLICKED,␣
↪(void *) "Single clicked");
    lv_obj_add_event_cb(btn, streak_event_cb, LV_EVENT_DOUBLE_CLICKED,␣
↪(void *) "Double clicked");
    lv_obj_add_event_cb(btn, streak_event_cb, LV_EVENT_TRIPLE_CLICKED,␣
↪(void *) "Triple clicked");

    lv_obj_t * label = lv_label_create(btn);
    lv_label_set_text(label, "Click me!");
    lv_obj_center(label);
}

#endif
```

Gestione di eventi multipli

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_SWITCH

static void event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * label = (lv_obj_t *) lv_event_get_user_data(e);

    switch(code) {
        case LV_EVENT_PRESSED:
            lv_label_set_text(label, "The last button event:\nLV_EVENT_
↪PRESSED");
            break;
        case LV_EVENT_CLICKED:
            lv_label_set_text(label, "The last button event:\nLV_EVENT_
↪CLICKED");
            break;
        case LV_EVENT_LONG_PRESSED:
            lv_label_set_text(label, "The last button event:\nLV_EVENT_
↪LONG_PRESSED");
            break;
        case LV_EVENT_LONG_PRESSED_REPEAT:
            lv_label_set_text(label, "The last button event:\nLV_EVENT_
↪LONG_PRESSED_REPEAT");
            break;
        default:
            break;
    }
}

/**
 * Handle multiple events
 */
void lv_example_event_button(void)
{
    lv_obj_t * btn = lv_button_create(lv_screen_active());
    lv_obj_set_size(btn, 100, 50);
    lv_obj_center(btn);

    lv_obj_t * btn_label = lv_label_create(btn);
    lv_label_set_text(btn_label, "Click me!");
    lv_obj_center(btn_label);

    lv_obj_t * info_label = lv_label_create(lv_screen_active());
    lv_label_set_text(info_label, "The last button event:\nNone");

    lv_obj_add_event_cb(btn, event_cb, LV_EVENT_ALL, info_label);
}

#endif
```

Evento bubbling

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_FLEX

static void event_cb(lv_event_t * e)
{
    /*The original target of the event. Can be the buttons or the␣
↪container*/
    lv_obj_t * target = lv_event_get_target_obj(e);

    /*The current target is always the container as the event is added to␣
↪it*/
    lv_obj_t * cont = lv_event_get_current_target_obj(e);

    /*If container was clicked do nothing*/
    if(target == cont) return;

    /*Make the clicked buttons red*/
    lv_obj_set_style_bg_color(target, lv_palette_main(LV_PALETTE_RED), 0);
}

/**
 * Demonstrate event bubbling
 */
void lv_example_event_bubble(void)
{

    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 290, 200);
    lv_obj_center(cont);
    lv_obj_set_flex_flow(cont, LV_FLEX_FLOW_ROW_WRAP);

    uint32_t i;
    for(i = 0; i < 30; i++) {
        lv_obj_t * btn = lv_button_create(cont);
        lv_obj_set_size(btn, 70, 50);
        lv_obj_add_flag(btn, LV_OBJ_FLAG_EVENT_BUBBLE);

        lv_obj_t * label = lv_label_create(btn);
        lv_label_set_text_fmt(label, "%" LV_PRIu32, i);
        lv_obj_center(label);
    }

    lv_obj_add_event_cb(cont, event_cb, LV_EVENT_CLICKED, NULL);
}

#endif
```

Event trickle-down

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_FLEX

/**
 * Demonstrate event trickle
 */
```

(continues on next page)

(continua dalla pagina precedente)

```
void lv_example_event_trickle(void)
{
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 290, 200);
    lv_obj_center(cont);
    lv_obj_set_flex_flow(cont, LV_FLEX_FLOW_ROW_WRAP);

    static lv_style_t style_black;
    lv_style_init(&style_black);
    lv_style_set_text_color(&style_black, lv_color_white());
    lv_style_set_bg_color(&style_black, lv_color_black());

    /*Enable event trickle-down on the container*/
    lv_obj_add_flag(cont, LV_OBJ_FLAG_EVENT_TRICKLE);

    lv_obj_add_style(cont, &style_black, LV_STATE_PRESSED);

    uint32_t i;
    for(i = 0; i < 9; i++) {
        lv_obj_t * subcont = lv_obj_create(cont);
        lv_obj_set_size(subcont, 70, 50);
        lv_obj_t * label = lv_label_create(subcont);
        lv_label_set_text_fmt(label, "%" LV_PRIu32, i);

        /*Add style to the label when clicked*/
        lv_obj_add_style(subcont, &style_black, LV_STATE_FOCUSED);
    }
}

#endif
```

Evento Draw

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

#if LV_BUILD_EXAMPLES

static int32_t size = 0;
static bool size_dec = false;

static void timer_cb(lv_timer_t * timer)
{
    lv_obj_invalidate((lv_obj_t *) lv_timer_get_user_data(timer));
    if(size_dec) size--;
    else size++;

    if(size == 50) size_dec = true;
    else if(size == 0) size_dec = false;
}

static void event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *) lv_draw_task_
↪get_draw_dsc(draw_task);
```

(continues on next page)

(continua dalla pagina precedente)

```
    if(base_dsc->part == LV_PART_MAIN) {
        lv_draw_rect_dsc_t draw_dsc;
        lv_draw_rect_dsc_init(&draw_dsc);
        draw_dsc.bg_color = lv_color_hex(0xffaaaa);
        draw_dsc.radius = LV_RADIUS_CIRCLE;
        draw_dsc.border_color = lv_color_hex(0xff5555);
        draw_dsc.border_width = 2;
        draw_dsc.outline_color = lv_color_hex(0xff0000);
        draw_dsc.outline_pad = 3;
        draw_dsc.outline_width = 2;

        lv_area_t a;
        a.x1 = 0;
        a.y1 = 0;
        a.x2 = size;
        a.y2 = size;
        lv_area_t obj_coords;
        lv_obj_get_coords(obj, &obj_coords);
        lv_area_align(&obj_coords, &a, LV_ALIGN_CENTER, 0, 0);

        lv_draw_rect(base_dsc->layer, &draw_dsc, &a);
    }
}

/**
 * Demonstrate the usage of draw event
 */
void lv_example_event_draw(void)
{
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 200, 200);
    lv_obj_center(cont);
    lv_obj_add_event_cb(cont, event_cb, LV_EVENT_DRAW_TASK_ADDED, NULL);
    lv_obj_add_flag(cont, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);
    lv_timer_create(timer_cb, 30, cont);
}

#endif
```

5.5.9 API

lv_display.h

lv_event.h

lv_event_private.h

lv_indev.h

lv_obj.h

lv_obj_event.h

lv_obj_event_private.h

5.6 Layout

5.6.1 Flex

Panoramica

The Flexbox (or Flex for short) is a subset of CSS Flexbox behaviors.

It can arrange items (child Widgets) into rows or columns (tracks), handle wrapping, adjust the spacing between items and tracks, handle *grow* to make item(s) fill remaining space with respect to min/max width and height.

To make a Widget a Flex container call `lv_obj_set_layout(widget, LV_LAYOUT_FLEX)`.

Note that the Flex layout feature of LVGL needs to be globally enabled with `LV_USE_FLEX` in `lv_conf.h`.

Terms

- **tracks**: rows or columns
- **main direction**: row or column, the direction in which multiple items are placed first
- **cross direction**: the direction perpendicular to the **main direction**
- **wrap**: if there is no more space in the track, a new track is started
- **grow**: if set on an item it will "grow" to fill the remaining space in the track. The available space will be distributed among items respective to their grow value (larger value means more space)
- **gap**: the space between rows and columns or the items on a track

See CSS Flexbox for illustrations showing the meanings of these terms.

Simple Interface

Use the following functions to set and control the Flex layout on any parent Widget.

> **Nota**
>
> The parent Widget must be a Flex container for these styles to take effect. The functions below cause the parent Widget to become a Flex container if it is not already.

Flex flow

`lv_obj_set_flex_flow(widget, flex_flow)`

The possible values for `flex_flow` are:

- `LV_FLEX_FLOW_ROW`: Place the children in a row without wrapping
- `LV_FLEX_FLOW_COLUMN`: Place the children in a column without wrapping
- `LV_FLEX_FLOW_ROW_WRAP`: Place the children in a row with wrapping
- `LV_FLEX_FLOW_COLUMN_WRAP`: Place the children in a column with wrapping
- `LV_FLEX_FLOW_ROW_REVERSE`: Place the children in a row without wrapping but in reversed order
- `LV_FLEX_FLOW_COLUMN_REVERSE`: Place the children in a column without wrapping but in reversed order
- `LV_FLEX_FLOW_ROW_WRAP_REVERSE`: Place the children in a row with wrapping but in reversed order
- `LV_FLEX_FLOW_COLUMN_WRAP_REVERSE`: Place the children in a column with wrapping but in reversed order

These values cause the Widget's layout behavior to model CSS Flexbox behavior by combining flex-direction and flex-wrap as defined under flex-flow.

Flex align

To manage placement of children use `lv_obj_set_flex_align(widget, main_place, cross_place, track_cross_place)` which makes the parent Widget model the Flex-container behavior defined here.

- `main_place` determines how to distribute the items in their track on the main axis. E.g. flush the items to the right on `LV_FLEX_FLOW_ROW_WRAP`. (It's called justify-content in CSS.)
- `cross_place` determines how to distribute the items in their track on the cross axis. E.g. if the items have different height, align them against the bottom of the track. (It's called align-items in CSS.)
- `track_cross_place` determines how to distribute the tracks (It's called align-content in CSS.)

The possible values are:

- `LV_FLEX_ALIGN_START`: means left when direction is horizontal, top when vertical (default)
- `LV_FLEX_ALIGN_END`: means right when direction is horizontal, bottom when vertical
- `LV_FLEX_ALIGN_CENTER`: simply center with respect to direction
- `LV_FLEX_ALIGN_SPACE_EVENLY`: items are distributed so that the spacing between any two items (and the space to the edges) is equal. Does not apply to `track_cross_place`.
- `LV_FLEX_ALIGN_SPACE_AROUND`: items are evenly distributed in the track with equal space around them. Note that visually the spaces are not equal since all the items have equal space on both sides. This makes the space between items double the space between edge items and the container's edge. Does not apply to `track_cross_place`.
- `LV_FLEX_ALIGN_SPACE_BETWEEN`: items are evenly distributed in the track with no space before and after first and last items. Does not apply to `track_cross_place`.

See justify-content, align-items and align-content for illustrations of these values.

Flex grow

Flex grow can be used to make one or more children fill available space in the track. When more than one child Widget have non-zero grow values, all available space will be distributed in proportion to their respective grow values. For example, if there is 400 px space remaining and 3 child Widgets with non-zero grow values:

- `A` with grow = 1
- `B` with grow = 1
- `C` with grow = 2

`A` and `B` will occupy 100 px, and `C` will occupy 200 px.

Flex grow can be set on a child Widget with `lv_obj_set_flex_grow(child, value)`. `value` needs to be >= 1 or 0 to disable grow on the child.

See flex-grow for an illustration of this behavior.

Style Interface

All Flex-related values are style properties under the hood so you can use them as you would any other style property.

The following flex related style properties exist:

- `FLEX_FLOW`
- `FLEX_MAIN_PLACE`
- `FLEX_CROSS_PLACE`
- `FLEX_TRACK_PLACE`
- `FLEX_GROW`

Internal padding

To modify the minimum space flexbox inserts between Widgets, the following functions can be used to set the flex container padding style:

- *lv_style_set_pad_row()* sets padding between rows.
- *lv_style_set_pad_column()* sets padding between columns.

These can, for example, be used if you do not want any padding between Widgets: *lv_style_set_pad_column*(&row_container_style, 0)

Other Features

RTL

If the base direction of the container is set the *LV_BASE_DIR_RTL* the meaning of *LV_FLEX_ALIGN_START* and *LV_FLEX_ALIGN_END* is swapped on ROW layouts. I.e. START will mean right.

The items on ROW layouts, and tracks of COLUMN layouts will be placed from right to left.

New track

You can force Flex to put an item into a new line with *lv_obj_add_flag*(child, *LV_OBJ_FLAG_FLEX_IN_NEW_TRACK*).

Further Reading

Learn more about CSS Flexbox.

Esempi

Un semplice layout di riga e colonna con flexbox

```
#include "../../lv_examples.h"
#if LV_USE_FLEX && LV_BUILD_EXAMPLES

/**
 * A simple row and a column layout with flexbox
 */
void lv_example_flex_1(void)
{
    /*Create a container with ROW flex direction*/
    lv_obj_t * cont_row = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont_row, 300, 75);
    lv_obj_align(cont_row, LV_ALIGN_TOP_MID, 0, 5);
    lv_obj_set_flex_flow(cont_row, LV_FLEX_FLOW_ROW);

    /*Create a container with COLUMN flex direction*/
    lv_obj_t * cont_col = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont_col, 200, 150);
    lv_obj_align_to(cont_col, cont_row, LV_ALIGN_OUT_BOTTOM_MID, 0, 5);
    lv_obj_set_flex_flow(cont_col, LV_FLEX_FLOW_COLUMN);

    uint32_t i;
    for(i = 0; i < 10; i++) {
        lv_obj_t * obj;
        lv_obj_t * label;
```

(continues on next page)

(continua dalla pagina precedente)

```
        /*Add items to the row*/
        obj = lv_button_create(cont_row);
        lv_obj_set_size(obj, 100, LV_PCT(100));

        label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "Item: %" LV_PRIu32"", i);
        lv_obj_center(label);

        /*Add items to the column*/
        obj = lv_button_create(cont_col);
        lv_obj_set_size(obj, LV_PCT(100), LV_SIZE_CONTENT);

        label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "Item: %" LV_PRIu32, i);
        lv_obj_center(label);
    }
}

#endif
```

Disporre gli elementi in righe con avvolgimento e spaziatura uniforme

```
#include "../../lv_examples.h"
#if LV_USE_FLEX && LV_BUILD_EXAMPLES

/**
 * Arrange items in rows with wrap and place the items to get even space␣
 ↪around them.
 */
void lv_example_flex_2(void)
{
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_flex_flow(&style, LV_FLEX_FLOW_ROW_WRAP);
    lv_style_set_flex_main_place(&style, LV_FLEX_ALIGN_SPACE_EVENLY);
    lv_style_set_layout(&style, LV_LAYOUT_FLEX);

    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);
    lv_obj_add_style(cont, &style, 0);

    uint32_t i;
    for(i = 0; i < 8; i++) {
        lv_obj_t * obj = lv_obj_create(cont);
        lv_obj_set_size(obj, 70, LV_SIZE_CONTENT);
        lv_obj_add_flag(obj, LV_OBJ_FLAG_CHECKABLE);

        lv_obj_t * label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "%" LV_PRIu32, i);
        lv_obj_center(label);
    }
}

#endif
```

Dimostrare la flex grow.

```
#include "../../lv_examples.h"
#if LV_USE_FLEX && LV_BUILD_EXAMPLES

/**
 * Demonstrate flex grow.
 */
void lv_example_flex_3(void)
{
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);
    lv_obj_set_flex_flow(cont, LV_FLEX_FLOW_ROW);

    lv_obj_t * obj;
    obj = lv_obj_create(cont);
    lv_obj_set_size(obj, 40, 40);           /*Fix size*/

    obj = lv_obj_create(cont);
    lv_obj_set_height(obj, 40);
    lv_obj_set_flex_grow(obj, 1);           /*1 portion from the free␣
↪space*/

    obj = lv_obj_create(cont);
    lv_obj_set_height(obj, 40);
    lv_obj_set_flex_grow(obj, 2);           /*2 portion from the free␣
↪space*/

    obj = lv_obj_create(cont);
    lv_obj_set_size(obj, 40, 40);           /*Fix size. It is flushed to␣
↪the right by the "grow" items*/
}

#endif
```

Dimostrare la flex grow.

```
#include "../../lv_examples.h"
#if LV_USE_FLEX && LV_BUILD_EXAMPLES

/**
 * Reverse the order of flex items
 */
void lv_example_flex_4(void)
{

    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);
    lv_obj_set_flex_flow(cont, LV_FLEX_FLOW_COLUMN_REVERSE);

    uint32_t i;
    for(i = 0; i < 6; i++) {
        lv_obj_t * obj = lv_obj_create(cont);
        lv_obj_set_size(obj, 100, 50);
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_obj_t * label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "Item: %" LV_PRIu32, i);
        lv_obj_center(label);
    }
}

#endif
```

Dimostrare le proprietà di stile degli spazi tra colonne e righe

```
#include "../../lv_examples.h"
#if LV_USE_FLEX && LV_BUILD_EXAMPLES

static void row_gap_anim(void * obj, int32_t v)
{
    lv_obj_set_style_pad_row((lv_obj_t *) obj, v, 0);
}

static void column_gap_anim(void * obj, int32_t v)
{
    lv_obj_set_style_pad_column((lv_obj_t *) obj, v, 0);
}

/**
 * Demonstrate the effect of column and row gap style properties
 */
void lv_example_flex_5(void)
{
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);
    lv_obj_set_flex_flow(cont, LV_FLEX_FLOW_ROW_WRAP);

    uint32_t i;
    for(i = 0; i < 9; i++) {
        lv_obj_t * obj = lv_obj_create(cont);
        lv_obj_set_size(obj, 70, LV_SIZE_CONTENT);

        lv_obj_t * label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "%" LV_PRIu32, i);
        lv_obj_center(label);
    }

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, cont);
    lv_anim_set_values(&a, 0, 10);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);

    lv_anim_set_exec_cb(&a, row_gap_anim);
    lv_anim_set_duration(&a, 500);
    lv_anim_set_reverse_duration(&a, 500);
    lv_anim_start(&a);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_anim_set_exec_cb(&a, column_gap_anim);
    lv_anim_set_duration(&a, 3000);
    lv_anim_set_reverse_duration(&a, 3000);
    lv_anim_start(&a);
}

#endif
```

La direzione di base RTL modifica l'ordine degli elementi

```
#include "../../lv_examples.h"
#if LV_USE_FLEX && LV_BUILD_EXAMPLES

/**
 * RTL base direction changes order of the items.
 * Also demonstrate how horizontal scrolling works with RTL.
 */
void lv_example_flex_6(void)
{
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_style_base_dir(cont, LV_BASE_DIR_RTL, 0);
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);
    lv_obj_set_flex_flow(cont, LV_FLEX_FLOW_ROW_WRAP);

    uint32_t i;
    for(i = 0; i < 20; i++) {
        lv_obj_t * obj = lv_obj_create(cont);
        lv_obj_set_size(obj, 70, LV_SIZE_CONTENT);

        lv_obj_t * label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "%" LV_PRIu32, i);
        lv_obj_center(label);
    }
}
#endif
```

API

lv_flex.h

5.6.2 Grid

Panoramica

The Grid layout is a subset of CSS Grid layout.

It can arrange items (child Widgets) into a 2D "table" that has rows and columns (tracks). An item can span multiple columns or rows. The track's size can be set in pixels, to the largest item (`LV_GRID_CONTENT`), or to a fraction of the available free space (i.e. Free [FR] Units) to distribute free space proportionally.

To make a Widget a Grid container call `lv_obj_set_layout(widget, LV_LAYOUT_GRID)`.

Note that the Grid layout feature of LVGL needs to be globally enabled with `LV_USE_GRID` in `lv_conf.h`.

Terms

- **tracks**: rows or columns
- **free (FR) units**: if a track's size is set in `FR units` it will grow to fill the remaining space in the parent Widget (container), in proportion with other tracks that have non-zero FR-unit values.
- **gap**: the space between rows and columns or the items on a track

Simple Interface

With the following functions you can cause any parent Widget to have Grid-layout behaviors.

> **Nota**
>
> As with Flex containers, the parent Widget must be a Grid container for these styles to take effect. The functions below cause the parent Widget to become a Grid container if it is not already.

Grid descriptors

First you need to describe the size of rows and columns. It can be done by declaring 2 arrays and the track sizes in them. The last element must be *LV_GRID_TEMPLATE_LAST*.

Ad esempio:

```
static int32_t column_dsc[] = {100, 400, LV_GRID_TEMPLATE_LAST};   /* 2␣
↪columns with 100- and 400-px width */
static int32_t row_dsc[] = {100, 100, 100, LV_GRID_TEMPLATE_LAST}; /* 3␣
↪100-px tall rows */
```

To set the descriptors on a parent use *lv_obj_set_grid_dsc_array*`(widget, col_dsc, row_dsc)`.

Besides settings the sizes in pixels, you can use two special values:

- *LV_GRID_CONTENT* sets size to fit the largest child on this track
- `LV_GRID_FR(X)` determines what portion of the remaining space should be used by this track. Larger values means larger space.

Grid items

By default, a Widget's children are not added to the grid. They need to be added manually to a cell.

To do this call *lv_obj_set_grid_cell*`(child, column_align, column_pos, column_span, row_align, row_pos, row_span)`.

`column_align` and `row_align` determine how to align the child Widget in its cell. Possible values are:

- *LV_GRID_ALIGN_START*: means left when direction is horizontal and top when vertical (default)
- *LV_GRID_ALIGN_END*: means right when direction is horizontal and bottom when vertical
- *LV_GRID_ALIGN_CENTER*: simply center `column_pos` and `row_pos` means the zero-based index of the cell in which the item should be placed.

`column_span` and `row_span` means how many tracks should be occupied from the start cell. Must be `>= 1`.

Grid align

If there is some empty space, items (Widgets) in Grid tracks can be aligned in several ways:

- *LV_GRID_ALIGN_START*: means left when direction is horizontal and top when vertical. (default)
- *LV_GRID_ALIGN_END*: means right when direction is horizontal and bottom when vertical

- *LV_GRID_ALIGN_CENTER*: simply center
- *LV_GRID_ALIGN_SPACE_EVENLY*: items are distributed so that the spacing between any two items (and the space to the edges) is equal. Not applies to `track_cross_place`.
- *LV_GRID_ALIGN_SPACE_AROUND*: items are evenly distributed in the track with equal space around them. Note that visually the spaces aren't equal, since all the items have equal space on both sides. This makes the space between items double the space between edge items and the container's edge. Does not apply to `track_cross_place`.
- *LV_GRID_ALIGN_SPACE_BETWEEN*: items are evenly distributed in the track with first and last items next to container's edges. Does not apply to `track_cross_place`.

To set the track's alignment use *lv_obj_set_grid_align*`(widget, column_align, row_align)`.

Sub grid

If you set the column and/or row grid descriptors of a widget to `NULL` it will use the grid descriptor(s) from it's parent.

For example if you create a grid item that spans columns 2..6 columns and rows 1..3 of the grid, the grid item will occupy 5 columns and 4 rows with the corresponding track size from the parent Grid container.

This way even if a wrapper item is used in the grid, it can be made "transparent" from the grid's point of view.

Limitations:

- The sub-grid is resolved only to a depth of 1 level. That is, a grid can have a sub-grid child, but that sub-grid cannot have another sub-grid.
- `LV_GRID_CONTENT` tracks on the grid are not handled in the sub-grid, only in its own grid.

The sub-grid feature works the same as in CSS. For further information, see CSS Subgrid.

Style Interface

All the Grid-related values are style properties under the hood so you can use them as you would any other style property.

The following Grid-related style properties exist:

- `GRID_COLUMN_DSC_ARRAY`
- `GRID_ROW_DSC_ARRAY`
- `GRID_COLUMN_ALIGN`
- `GRID_ROW_ALIGN`
- `GRID_CELL_X_ALIGN`
- `GRID_CELL_COLUMN_POS`
- `GRID_CELL_COLUMN_SPAN`
- `GRID_CELL_Y_ALIGN`
- `GRID_CELL_ROW_POS`
- `GRID_CELL_ROW_SPAN`

Internal padding

To modify the minimum space Grid inserts between Widgets, the following properties can be set on the Grid container style:

- *lv_style_set_pad_row()* sets padding between rows.
- *lv_style_set_pad_column()* sets padding between columns.

Other features

RTL

If the base direction of the container is set to *LV_BASE_DIR_RTL*, the meaning of *LV_GRID_ALIGN_START* and *LV_GRID_ALIGN_END* is swapped. I.e. START will mean right-most.

The columns will be placed from right to left.

Further Reading

- Learn more about CSS Grid layout.
- Learn more about CSS Subgrid layout.

Esempio

Una semplice grid

```
#include "../../lv_examples.h"
#if LV_USE_GRID && LV_BUILD_EXAMPLES

/**
 * A simple grid
 */
void lv_example_grid_1(void)
{
    static int32_t col_dsc[] = {70, 70, 70, LV_GRID_TEMPLATE_LAST};
    static int32_t row_dsc[] = {50, 50, 50, LV_GRID_TEMPLATE_LAST};

    /*Create a container with grid*/
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_style_grid_column_dsc_array(cont, col_dsc, 0);
    lv_obj_set_style_grid_row_dsc_array(cont, row_dsc, 0);
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);
    lv_obj_set_layout(cont, LV_LAYOUT_GRID);

    lv_obj_t * label;
    lv_obj_t * obj;

    uint8_t i;
    for(i = 0; i < 9; i++) {
        uint8_t col = i % 3;
        uint8_t row = i / 3;

        obj = lv_button_create(cont);
        /*Stretch the cell horizontally and vertically too
         *Set span to 1 to make the cell 1 column/row sized*/
        lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_STRETCH, col, 1,
                             LV_GRID_ALIGN_STRETCH, row, 1);

        label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "c%d, r%d", col, row);
        lv_obj_center(label);
    }
}
```

(continues on next page)

(continua dalla pagina precedente)

```
#endif
```

Posizionamento e ampiezza delle celle

```
#include "../../lv_examples.h"
#if LV_USE_GRID && LV_BUILD_EXAMPLES

/**
 * Demonstrate cell placement and span
 */
void lv_example_grid_2(void)
{
    static int32_t col_dsc[] = {70, 70, 70, LV_GRID_TEMPLATE_LAST};
    static int32_t row_dsc[] = {50, 50, 50, LV_GRID_TEMPLATE_LAST};

    /*Create a container with grid*/
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_grid_dsc_array(cont, col_dsc, row_dsc);
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);

    lv_obj_t * label;
    lv_obj_t * obj;

    /*Cell to 0;0 and align to to the start (left/top) horizontally and␣
↪vertically too*/
    obj = lv_obj_create(cont);
    lv_obj_set_size(obj, LV_SIZE_CONTENT, LV_SIZE_CONTENT);
    lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_START, 0, 1,
                         LV_GRID_ALIGN_START, 0, 1);
    label = lv_label_create(obj);
    lv_label_set_text(label, "c0, r0");

    /*Cell to 1;0 and align to to the start (left) horizontally and center␣
↪vertically too*/
    obj = lv_obj_create(cont);
    lv_obj_set_size(obj, LV_SIZE_CONTENT, LV_SIZE_CONTENT);
    lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_START, 1, 1,
                         LV_GRID_ALIGN_CENTER, 0, 1);
    label = lv_label_create(obj);
    lv_label_set_text(label, "c1, r0");

    /*Cell to 2;0 and align to to the start (left) horizontally and end␣
↪(bottom) vertically too*/
    obj = lv_obj_create(cont);
    lv_obj_set_size(obj, LV_SIZE_CONTENT, LV_SIZE_CONTENT);
    lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_START, 2, 1,
                         LV_GRID_ALIGN_END, 0, 1);
    label = lv_label_create(obj);
    lv_label_set_text(label, "c2, r0");

    /*Cell to 1;1 but 2 column wide (span = 2).Set width and height to␣
↪stretched.*/
    obj = lv_obj_create(cont);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_size(obj, LV_SIZE_CONTENT, LV_SIZE_CONTENT);
    lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_STRETCH, 1, 2,
                         LV_GRID_ALIGN_STRETCH, 1, 1);
    label = lv_label_create(obj);
    lv_label_set_text(label, "c1-2, r1");

    /*Cell to 0;1 but 2 rows tall (span = 2).Set width and height to␣
↪stretched.*/
    obj = lv_obj_create(cont);
    lv_obj_set_size(obj, LV_SIZE_CONTENT, LV_SIZE_CONTENT);
    lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_STRETCH, 0, 1,
                         LV_GRID_ALIGN_STRETCH, 1, 2);
    label = lv_label_create(obj);
    lv_label_set_text(label, "c0\nr1-2");
}

#endif
```

La -free unit- della grid

```
#include "../../lv_examples.h"
#if LV_USE_GRID && LV_BUILD_EXAMPLES

/**
 * Demonstrate grid's "free unit"
 */
void lv_example_grid_3(void)
{
    /*Column 1: fix width 60 px
     *Column 2: 1 unit from the remaining free space
     *Column 3: 2 unit from the remaining free space*/
    static int32_t col_dsc[] = {60, LV_GRID_FR(1), LV_GRID_FR(2), LV_GRID_
↪TEMPLATE_LAST};

    /*Row 1: fix width 50 px
     *Row 2: 1 unit from the remaining free space
     *Row 3: fix width 50 px*/
    static int32_t row_dsc[] = {50, LV_GRID_FR(1), 50, LV_GRID_TEMPLATE_
↪LAST};

    /*Create a container with grid*/
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);
    lv_obj_set_grid_dsc_array(cont, col_dsc, row_dsc);

    lv_obj_t * label;
    lv_obj_t * obj;
    uint8_t i;
    for(i = 0; i < 9; i++) {
        uint8_t col = i % 3;
        uint8_t row = i / 3;

        obj = lv_obj_create(cont);
        /*Stretch the cell horizontally and vertically too
```

(continues on next page)

(continua dalla pagina precedente)

```
         *Set span to 1 to make the cell 1 column/row sized*/
        lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_STRETCH, col, 1,
                             LV_GRID_ALIGN_STRETCH, row, 1);

        label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "%d,%d", col, row);
        lv_obj_center(label);
    }
}

#endif
```

Posizionamento delle tracce

```
#include "../../lv_examples.h"
#if LV_USE_GRID && LV_BUILD_EXAMPLES

/**
 * Demonstrate track placement
 */
void lv_example_grid_4(void)
{
    static int32_t col_dsc[] = {60, 60, 60, LV_GRID_TEMPLATE_LAST};
    static int32_t row_dsc[] = {45, 45, 45, LV_GRID_TEMPLATE_LAST};

    /*Add space between the columns and move the rows to the bottom (end)*/

    /*Create a container with grid*/
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_grid_align(cont, LV_GRID_ALIGN_SPACE_BETWEEN, LV_GRID_ALIGN_
↪END);
    lv_obj_set_grid_dsc_array(cont, col_dsc, row_dsc);
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);

    lv_obj_t * label;
    lv_obj_t * obj;
    uint8_t i;
    for(i = 0; i < 9; i++) {
        uint8_t col = i % 3;
        uint8_t row = i / 3;

        obj = lv_obj_create(cont);
        /*Stretch the cell horizontally and vertically too
         *Set span to 1 to make the cell 1 column/row sized*/
        lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_STRETCH, col, 1,
                             LV_GRID_ALIGN_STRETCH, row, 1);

        label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "%d,%d", col, row);
        lv_obj_center(label);
    }
}

#endif
```

Spazi tra colonne e righe

```
#include "../../lv_examples.h"
#if LV_USE_GRID && LV_BUILD_EXAMPLES

static void row_gap_anim(void * obj, int32_t v)
{
    lv_obj_set_style_pad_row((lv_obj_t *) obj, v, 0);
}

static void column_gap_anim(void * obj, int32_t v)
{
    lv_obj_set_style_pad_column((lv_obj_t *) obj, v, 0);
}

/**
 * Demonstrate column and row gap
 */
void lv_example_grid_5(void)
{

    /*60x60 cells*/
    static int32_t col_dsc[] = {60, 60, 60, LV_GRID_TEMPLATE_LAST};
    static int32_t row_dsc[] = {45, 45, 45, LV_GRID_TEMPLATE_LAST};

    /*Create a container with grid*/
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);
    lv_obj_set_grid_dsc_array(cont, col_dsc, row_dsc);

    lv_obj_t * label;
    lv_obj_t * obj;
    uint8_t i;
    for(i = 0; i < 9; i++) {
        uint8_t col = i % 3;
        uint8_t row = i / 3;

        obj = lv_obj_create(cont);
        lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_STRETCH, col, 1,
                             LV_GRID_ALIGN_STRETCH, row, 1);
        label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "%d,%d", col, row);
        lv_obj_center(label);
    }

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, cont);
    lv_anim_set_values(&a, 0, 10);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);

    lv_anim_set_exec_cb(&a, row_gap_anim);
    lv_anim_set_duration(&a, 500);
    lv_anim_set_reverse_duration(&a, 500);
    lv_anim_start(&a);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_anim_set_exec_cb(&a, column_gap_anim);
    lv_anim_set_duration(&a, 3000);
    lv_anim_set_reverse_duration(&a, 3000);
    lv_anim_start(&a);
}

#endif
```

Direzione RTL sulla griglia

```
#include "../../lv_examples.h"
#if LV_USE_GRID && LV_BUILD_EXAMPLES

/**
 * Demonstrate RTL direction on grid
 */
void lv_example_grid_6(void)
{

    static int32_t col_dsc[] = {60, 60, 60, LV_GRID_TEMPLATE_LAST};
    static int32_t row_dsc[] = {45, 45, 45, LV_GRID_TEMPLATE_LAST};

    /*Create a container with grid*/
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 300, 220);
    lv_obj_center(cont);
    lv_obj_set_style_base_dir(cont, LV_BASE_DIR_RTL, 0);
    lv_obj_set_grid_dsc_array(cont, col_dsc, row_dsc);

    lv_obj_t * label;
    lv_obj_t * obj;
    uint8_t i;
    for(i = 0; i < 9; i++) {
        uint8_t col = i % 3;
        uint8_t row = i / 3;

        obj = lv_obj_create(cont);
        /*Stretch the cell horizontally and vertically too
         *Set span to 1 to make the cell 1 column/row sized*/
        lv_obj_set_grid_cell(obj, LV_GRID_ALIGN_STRETCH, col, 1,
                             LV_GRID_ALIGN_STRETCH, row, 1);

        label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "%d,%d", col, row);
        lv_obj_center(label);
    }
}

#endif
```

API

lv_grid.h

lv_gridnav.h

5.7 Scrolling

5.7.1 Panoramica

In LVGL scrolling works very intuitively: if a Widget is outside its parent content area (the size without padding), the parent becomes scrollable and scrollbar(s) will appear. That's it.

Any Widget can be scrollable including *Base Widget (lv_obj)*, `lv_image`, `lv_button`, `lv_meter`, etc

The Widget can either be scrolled horizontally or vertically in one stroke; diagonal scrolling is not possible.

Scrollbar

Mode

Scrollbars are displayed according to the configured `scrollbar-mode`. The following modes are available:

- *LV_SCROLLBAR_MODE_OFF*: Never show the scrollbars
- *LV_SCROLLBAR_MODE_ON*: Always show the scrollbars
- *LV_SCROLLBAR_MODE_ACTIVE*: Show scroll bars while a Widget is being scrolled
- *LV_SCROLLBAR_MODE_AUTO*: Show scroll bars when the content is large enough to be scrolled

lv_obj_set_scrollbar_mode`(widget, LV_SCROLLBAR_MODE_...)` sets the scrollbar mode on a Widget.

Styling

A Scrollbar is a dedicated part of a Widget, called *LV_PART_SCROLLBAR*. For example, a scrollbar can turn to red like this:

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

...

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

A Widget goes to the *LV_STATE_SCROLLED* state while it's being scrolled. This allows adding different styles to the Widget that will be effective while it is being scrolled. For example, this code makes the scrollbar blue while the Widget is being scrolled:

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

...

lv_obj_add_style(widget, &style_blue, LV_STATE_SCROLLED | LV_PART_
↪SCROLLBAR);
```

If the base direction of the *LV_PART_SCROLLBAR* is RTL (`LV_BASE_DIR_RTL`) the vertical scrollbar will be placed on the left. Note that, the `base_dir` style property is inherited. Therefore, it can be set directly on the *LV_PART_SCROLLBAR* part of a Widget, or on the Widget's LV_PART_MAIN part, or that of any of its parents, to make a scrollbar inherit the base direction.

`pad_left/right/top/bottom` sets the spacing around the scrollbars, `width` sets the scrollbar's width and `length` sets the scrollbar's length: If `length` is not set or left at `0` the scrollbar's length will be set automatically according to the length of the content. .. code-block:: c

static lv_style_t style_scrollbar; lv_style_init(&style_scrollbar); lv_style_set_pad_left(&style_scrollbar, 2); lv_style_set_pad_right(&style_scrollbar, 2); lv_style_set_pad_top(&style_scrollbar, 2); lv_style_set_pad_bottom(&style_scrollbar, 2); lv_style_set_width(&style_scrollbar, 10); lv_style_set_length(&style_scrollbar, 50);

...

lv_obj_add_style(widget, &style_scrollbar, LV_PART_SCROLLBAR);

The minimum length of the scrollbar is fixed to 10, while its maximum length is limited by the Widget's height or width, depending on whether the scrollbar is vertical or horizontal. Any length value set outside of these limits will automatically result in a length fixed to either limit.

Scrolling Events

The following events are emitted as part of scrolling:

- *LV_EVENT_SCROLL_BEGIN*: Signals that scrolling has begun. The event parameter is `NULL` or an `lv_anim_t *` with a scroll animation descriptor that can be modified if required.
- *LV_EVENT_SCROLL_END*: Signals that scrolling has ended.
- *LV_EVENT_SCROLL*: Signals that the scrolling position changed; triggered on every position change.

5.7.2 Features of Scrolling

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

Scrollable

It is possible to make a Widget non-scrollable with *lv_obj_remove_flag*`(widget, `*LV_OBJ_FLAG_SCROLLABLE*`)`.

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

The direction in which scrolling happens can be controlled by *lv_obj_set_scroll_dir*`(widget, LV_DIR_...)`.

The following values can be used for the direction:

- *LV_DIR_TOP*: only scroll up
- *LV_DIR_LEFT*: only scroll left
- *LV_DIR_BOTTOM*: only scroll down
- *LV_DIR_RIGHT*: only scroll right
- *LV_DIR_HOR*: only scroll horizontally
- *LV_DIR_VER*: only scroll vertically
- *LV_DIR_ALL*: scroll any directions

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

Scroll chaining

If a Widget can't be scrolled further (e.g. its content has reached the bottom-most position), additional scrolling is propagated to its parent. If the parent can be scrolled in that direction than it will be scrolled instead. It continues propagating up the Widget's parent hierarchy up to the *Screen*.

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

Scroll momentum

When the user scrolls a Widget and releases it, LVGL can emulate inertial momentum for the scrolling. It's like the Widget was "thrown" and scrolling slows down smoothly.

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

Elastic scroll

Normally a Widget can't be scrolled past the extremities of its content. That is, the top side of the content can't be below the top side of the Widget, and vice versa for the bottom side.

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

Aggancio

The children of a Widget can be snapped according to specific rules when scrolling ends. Children can be made snappable individually with the `LV_OBJ_FLAG_SNAPPABLE` flag.

A Widget can align snapped children in four ways:

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

Snap alignment is set with `lv_obj_set_scroll_snap_x(widget, LV_SCROLL_SNAP_...)` and `lv_obj_set_scroll_snap_y(widget, LV_SCROLL_SNAP_...)`.

This is what happens under the hood:

1. user scrolls and releases a Widget;
2. LVGL calculates where the scroll would end considering scroll momentum;
3. LVGL finds the nearest scroll point;
4. LVGL scrolls to the snap point with an animation.

Scroll one

The "scroll one" feature tells LVGL to allow scrolling only one snappable child at a time. This requires making the children snappable and setting scroll snap alignment to something other than `LV_SCROLL_SNAP_NONE`.

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

Scroll on focus

Imagine that there are a lot of Widgets in a group that are on a scrollable Widget. Pressing the "Tab" button moves focus to the next Widget but it might be outside the visible area of the scrollable Widget. If the "scroll on focus" feature is enabled LVGL will automatically scroll Widgets to bring the child Widget with focus into view. The scrolling happens recursively therefore even nested scrollable Widgets are handled properly. The Widget will be scrolled into view even if it is on a different page of a tabview.

5.7.3 Scrolling Programmatically

The following API functions allow programmatic scrolling of Widgets:

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

- `lv_obj_scroll_to_x(widget, x, LV_ANIM_ON/OFF)` scroll to bring the given coordinate to the left side
- `lv_obj_scroll_to_y(widget, y, LV_ANIM_ON/OFF)` scroll to bring the given coordinate to the top side

From time to time you may need to retrieve the *scroll position* of a scrollable Widget, either to restore it later, or to dynamically display some elements according to its current scroll position. Here is an example to illustrate how to combine scroll event and store the scroll-top position.

```
static int scroll_value = 0;

static void store_scroll_top_value_event_cb(lv_event_t* e) {
  lv_obj_t * scr = lv_event_get_target(e);
  scroll_value = lv_obj_get_scroll_top(scr);
  printf("%d pixels are scrolled above top edge of display.\n", scroll_
↪value);
}

lv_obj_t * scr = lv_obj_create(NULL);
lv_obj_add_event_cb(scr, store_scroll_top_value_event_cb, LV_EVENT_SCROLL,␣
↪NULL);
```

Scroll coordinates can be retrieved from different axes with these functions:

- *`lv_obj_get_scroll_x`*`(widget)` Pixels scrolled past left edge of Widget's view window.
- *`lv_obj_get_scroll_y`*`(widget)` Pixels scrolled past top of Widget's view window.
- *`lv_obj_get_scroll_top`*`(widget)` Identical to *`lv_obj_get_scroll_y`*`(widget)`
- *`lv_obj_get_scroll_bottom`*`(widget)` Pixels scrolled past bottom of Widget's view window.
- *`lv_obj_get_scroll_left`*`(widget)` Identical to *`lv_obj_get_scroll_x`*`(widget)`.
- *`lv_obj_get_scroll_right`*`(widget)` Pixels scrolled past right edge of Widget's view window.

Setting scroll position can be done with these functions:

- *`lv_obj_scroll_by`*`(widget, dx, dy, anim_enable)` Scroll by given amount of pixels.
- *`lv_obj_scroll_by_bounded`*`(widget, dx, dy, animation_enable)` Scroll by given amount of pixels.
- *`lv_obj_scroll_to`*`(widget, x, y, animation_enable)` Scroll to given coordinate on Widget.
- *`lv_obj_scroll_to_x`*`(widget, x, animation_enable)` Scroll to X coordinate on Widget.
- *`lv_obj_scroll_to_y`*`(widget, y, animation_enable)` Scroll to Y coordinate on Widget.
- *`lv_obj_scroll_to_view`*`(widget, animation_enable)` Scroll `obj`'s parent Widget until `obj` becomes visible.
- *`lv_obj_scroll_to_view_recursive`*`(widget, animation_enable)` Scroll `obj`'s parent Widgets recursively until `obj` becomes visible.

5.7.4 Self Size

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

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

This means not only the children can make a Widget scrollable but a larger self size will as well.

LVGL uses the *LV_EVENT_GET_SELF_SIZE* event to get the self size of a Widget. Here is an example to see how to handle the event:

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

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

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

5.7.5 Esempi

Scrolling nidificato

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

static lv_obj_t  * panel;
static lv_obj_t  * save_button;
static lv_obj_t  * restore_button;
static int         saved_scroll_x;
static int         saved_scroll_y;

static void scroll_update_cb(lv_event_t * e);
static void button_event_cb(lv_event_t * e);

static void scroll_update_cb(lv_event_t * e)
{
    LV_UNUSED(e);

    LV_LOG("scroll info: x:%3" LV_PRId32", y:%3" LV_PRId32", top:%3" LV_
↪PRId32", "
           "bottom:%3" LV_PRId32", left:%3" LV_PRId32", right:%3" LV_PRId32
↪"\n",
           lv_obj_get_scroll_x(panel),
           lv_obj_get_scroll_y(panel),
           lv_obj_get_scroll_top(panel),
           lv_obj_get_scroll_bottom(panel),
           lv_obj_get_scroll_left(panel),
           lv_obj_get_scroll_right(panel)
          );
}

static void button_event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);

    if(obj == save_button) {
        saved_scroll_x = lv_obj_get_scroll_x(panel);
        saved_scroll_y = lv_obj_get_scroll_y(panel);
```

(continues on next page)

(continua dalla pagina precedente)

```
    }
    else {
        lv_obj_scroll_to(panel, saved_scroll_x, saved_scroll_y, LV_ANIM_
↪ON);
    }
}

/**
 * Demonstrate how scrolling appears automatically
 */
void lv_example_scroll_1(void)
{
    /*Create an object with the new style*/
    lv_obj_t * scr;
    scr = lv_screen_active();
    panel = lv_obj_create(scr);
    lv_obj_set_size(panel, 200, 200);
    lv_obj_align(panel, LV_ALIGN_CENTER, 44, 0);

    lv_obj_t * child;
    lv_obj_t * label;

    child = lv_obj_create(panel);
    lv_obj_set_pos(child, 0, 0);
    lv_obj_set_size(child, 70, 70);
    label = lv_label_create(child);
    lv_label_set_text(label, "Zero");
    lv_obj_center(label);

    child = lv_obj_create(panel);
    lv_obj_set_pos(child, 160, 80);
    lv_obj_set_size(child, 80, 80);

    lv_obj_t * child2 = lv_button_create(child);
    lv_obj_set_size(child2, 100, 50);

    label = lv_label_create(child2);
    lv_label_set_text(label, "Right");
    lv_obj_center(label);

    child = lv_obj_create(panel);
    lv_obj_set_pos(child, 40, 160);
    lv_obj_set_size(child, 100, 70);
    label = lv_label_create(child);
    lv_label_set_text(label, "Bottom");
    lv_obj_center(label);

    /* When LV_OBJ_FLAG_SCROLL_ELASTIC is cleared, scrolling does not go␣
↪past edge boundaries. */
    /* lv_obj_clear_flag(panel, LV_OBJ_FLAG_SCROLL_ELASTIC); */

    /* Call `scroll_update_cb` while panel is being scrolled. */
    lv_obj_add_event_cb(panel, scroll_update_cb, LV_EVENT_SCROLL, NULL);

    /* Set up buttons that save and restore scroll position. */
    save_button = lv_button_create(scr);
```

(continues on next page)

(continua dalla pagina precedente)

```
    restore_button = lv_button_create(scr);
    lv_obj_t * lbl;
    lbl = lv_label_create(save_button);
    lv_label_set_text_static(lbl, "Save");
    lbl = lv_label_create(restore_button);
    lv_label_set_text_static(lbl, "Restore");
    lv_obj_align_to(save_button, panel, LV_ALIGN_OUT_LEFT_MID, -10, -20);
    lv_obj_align_to(restore_button, panel, LV_ALIGN_OUT_LEFT_MID, -10, 20);
    lv_obj_add_event_cb(save_button, button_event_cb, LV_EVENT_CLICKED,␣
↪NULL);
    lv_obj_add_event_cb(restore_button, button_event_cb, LV_EVENT_CLICKED,␣
↪NULL);
}

#endif
```

Aggancio

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_FLEX

static void sw_event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * sw = lv_event_get_target_obj(e);

    if(code == LV_EVENT_VALUE_CHANGED) {
        lv_obj_t * list = (lv_obj_t *) lv_event_get_user_data(e);

        if(lv_obj_has_state(sw, LV_STATE_CHECKED)) lv_obj_add_flag(list,␣
↪LV_OBJ_FLAG_SCROLL_ONE);
        else lv_obj_remove_flag(list, LV_OBJ_FLAG_SCROLL_ONE);
    }
}

/**
 * Show an example to scroll snap
 */
void lv_example_scroll_2(void)
{
    lv_obj_t * panel = lv_obj_create(lv_screen_active());
    lv_obj_set_size(panel, 280, 120);
    lv_obj_set_scroll_snap_x(panel, LV_SCROLL_SNAP_CENTER);
    lv_obj_set_flex_flow(panel, LV_FLEX_FLOW_ROW);
    lv_obj_align(panel, LV_ALIGN_CENTER, 0, 20);

    uint32_t i;
    for(i = 0; i < 10; i++) {
        lv_obj_t * btn = lv_button_create(panel);
        lv_obj_set_size(btn, 150, lv_pct(100));

        lv_obj_t * label = lv_label_create(btn);
        if(i == 3) {
            lv_label_set_text_fmt(label, "Panel %" LV_PRIu32"\nno snap",␣
↪i);
            lv_obj_remove_flag(btn, LV_OBJ_FLAG_SNAPPABLE);
```

(continues on next page)

(continua dalla pagina precedente)

```
        }
        else {
            lv_label_set_text_fmt(label, "Panel %" LV_PRIu32, i);
        }

        lv_obj_center(label);
    }
    lv_obj_update_snap(panel, LV_ANIM_ON);

#if LV_USE_SWITCH
    /*Switch between "One scroll" and "Normal scroll" mode*/
    lv_obj_t * sw = lv_switch_create(lv_screen_active());
    lv_obj_align(sw, LV_ALIGN_TOP_RIGHT, -20, 10);
    lv_obj_add_event_cb(sw, sw_event_cb, LV_EVENT_ALL, panel);
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "One scroll");
    lv_obj_align_to(label, sw, LV_ALIGN_OUT_BOTTOM_MID, 0, 5);
#endif
}

#endif
```

Pulsante mobile

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_LIST

static uint32_t btn_cnt = 1;

static void float_button_event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * float_btn = lv_event_get_target_obj(e);

    if(code == LV_EVENT_CLICKED) {
        lv_obj_t * list = (lv_obj_t *) lv_event_get_user_data(e);
        char buf[32];
        lv_snprintf(buf, sizeof(buf), "Track %d", (int)btn_cnt);
        lv_obj_t * list_btn = lv_list_add_button(list, LV_SYMBOL_AUDIO,␣
↪buf);
        btn_cnt++;

        lv_obj_move_foreground(float_btn);

        lv_obj_scroll_to_view(list_btn, LV_ANIM_ON);
    }
}

/**
 * Create a list with a floating button
 */
void lv_example_scroll_3(void)
{
    lv_obj_t * list = lv_list_create(lv_screen_active());
    lv_obj_set_size(list, 280, 220);
    lv_obj_center(list);
```

(continues on next page)

(continua dalla pagina precedente)

```
    for(btn_cnt = 1; btn_cnt <= 2; btn_cnt++) {
        char buf[32];
        lv_snprintf(buf, sizeof(buf), "Track %d", (int)btn_cnt);
        lv_list_add_button(list, LV_SYMBOL_AUDIO, buf);
    }

    lv_obj_t * float_btn = lv_button_create(list);
    lv_obj_set_size(float_btn, 50, 50);
    lv_obj_add_flag(float_btn, LV_OBJ_FLAG_FLOATING);
    lv_obj_align(float_btn, LV_ALIGN_BOTTOM_RIGHT, 0, -lv_obj_get_style_
↪pad_right(list, LV_PART_MAIN));
    lv_obj_add_event_cb(float_btn, float_button_event_cb, LV_EVENT_ALL,␣
↪list);
    lv_obj_set_style_radius(float_btn, LV_RADIUS_CIRCLE, 0);
    lv_obj_set_style_bg_image_src(float_btn, LV_SYMBOL_PLUS, 0);
    lv_obj_set_style_text_font(float_btn, lv_theme_get_font_large(float_
↪btn), 0);
}

#endif
```

Stile delle barre di scorrimento

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_LIST

/**
 * Styling the scrollbars
 */
void lv_example_scroll_4(void)
{
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_set_size(obj, 200, 100);
    lv_obj_center(obj);

    lv_obj_t * label = lv_label_create(obj);
    lv_label_set_text(label,
                      "Lorem ipsum dolor sit amet, consectetur adipiscing␣
↪elit.\n"
                      "Etiam dictum, tortor vestibulum lacinia laoreet, mi␣
↪neque consectetur neque, vel mattis odio dolor egestas ligula. \n"
                      "Sed vestibulum sapien nulla, id convallis ex␣
↪porttitor nec. \n"
                      "Duis et massa eu libero accumsan faucibus a in arcu.
↪ \n"
                      "Ut pulvinar odio lorem, vel tempus turpis␣
↪condimentum quis. Nam consectetur condimentum sem in auctor. \n"
                      "Sed nisl augue, venenatis in blandit et, gravida ac␣
↪tortor. \n"
                      "Etiam dapibus elementum suscipit. \n"
                      "Proin mollis sollicitudin convallis. \n"
                      "Integer dapibus tempus arcu nec viverra. \n"
                      "Donec molestie nulla enim, eu interdum velit␣
↪placerat quis. \n"
                      "Donec id efficitur risus, at molestie turpis. \n"
```

(continues on next page)

(continua dalla pagina precedente)

```
                        "Suspendisse vestibulum consectetur nunc ut commodo.␣
↪\n"
                        "Fusce molestie rhoncus nisi sit amet tincidunt. \n"
                        "Suspendisse a nunc ut magna ornare volutpat.");

    /*Remove the style of scrollbar to have clean start*/
    lv_obj_remove_style(obj, NULL, LV_PART_SCROLLBAR | LV_STATE_ANY);

    /*Create a transition the animate the some properties on state change*/
    static const lv_style_prop_t props[] = {LV_STYLE_BG_OPA, LV_STYLE_
↪WIDTH, 0};
    static lv_style_transition_dsc_t trans;
    lv_style_transition_dsc_init(&trans, props, lv_anim_path_linear, 200,␣
↪0, NULL);

    /*Create a style for the scrollbars*/
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_width(&style, 4);      /*Width of the scrollbar*/
    lv_style_set_length(&style, 20);    /*Length of the scrollbar*/
    lv_style_set_pad_right(&style, 5);  /*Space from the parallel side*/
    lv_style_set_pad_top(&style, 5);    /*Space from the perpendicular␣
↪side*/

    lv_style_set_radius(&style, 2);
    lv_style_set_bg_opa(&style, LV_OPA_70);
    lv_style_set_bg_color(&style, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_border_color(&style, lv_palette_darken(LV_PALETTE_BLUE,␣
↪3));
    lv_style_set_border_width(&style, 2);
    lv_style_set_shadow_width(&style, 8);
    lv_style_set_shadow_spread(&style, 2);
    lv_style_set_shadow_color(&style, lv_palette_darken(LV_PALETTE_BLUE,␣
↪1));

    lv_style_set_transition(&style, &trans);

    /*Make the scrollbars wider and use 100% opacity when scrolled*/
    static lv_style_t style_scrolled;
    lv_style_init(&style_scrolled);
    lv_style_set_width(&style_scrolled, 8);
    lv_style_set_bg_opa(&style_scrolled, LV_OPA_COVER);

    lv_obj_add_style(obj, &style, LV_PART_SCROLLBAR);
    lv_obj_add_style(obj, &style_scrolled, LV_PART_SCROLLBAR | LV_STATE_
↪SCROLLED);
}

#endif
```

Scorrimento da destra a sinistra

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_FONT_DEJAVU_16_PERSIAN_HEBREW

/**
```

(continues on next page)

(continua dalla pagina precedente)

```
 * Scrolling with Right To Left base direction
 */
void lv_example_scroll_5(void)
{
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_set_style_base_dir(obj, LV_BASE_DIR_RTL, 0);
    lv_obj_set_size(obj, 200, 100);
    lv_obj_center(obj);

    lv_obj_t * label = lv_label_create(obj);
    lv_label_set_text(label,
                      "میکروکُنترولر (به انگلیسی: Microcontroller) گونه‌ای␣
↪ریزپردازنده است که دارای حافظهٔ دسترسی تصادفی (RAM) و حافظهٔ فقط‌خواندنی␣
↪(ROM)، تایمر، پورت‌های ورودی و خروجی (I/O) و درگاه ترتیبی (Serial Port␣
↪پورت سریال)، درون خود تراشه است، و می‌تواند به تنهایی ابزارهای دیگر را␣
↪کنترل کند. به عبارت دیگر یک میکروکنترلر، مدار مجتمع کوچکی است که از یک␣
↪CPU کوچک و اجزای دیگری مانند تایمر، درگاه‌های ورودی و خروجی آنالوگ و␣
↪دیجیتال و حافظه تشکیل شده‌است.");
    lv_obj_set_width(label, 400);
    lv_obj_set_style_text_font(label, &lv_font_dejavu_16_persian_hebrew,␣
↪0);

}

#endif
```

Traslazione durante lo scorrimento

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_FLEX

static void scroll_event_cb(lv_event_t * e)
{
    lv_obj_t * cont = lv_event_get_target_obj(e);

    lv_area_t cont_a;
    lv_obj_get_coords(cont, &cont_a);
    int32_t cont_y_center = cont_a.y1 + lv_area_get_height(&cont_a) / 2;

    int32_t r = lv_obj_get_height(cont) * 7 / 10;
    int32_t i;
    int32_t child_cnt = (int32_t)lv_obj_get_child_count(cont);
    for(i = 0; i < child_cnt; i++) {
        lv_obj_t * child = lv_obj_get_child(cont, i);
        lv_area_t child_a;
        lv_obj_get_coords(child, &child_a);

        int32_t child_y_center = child_a.y1 + lv_area_get_height(&child_a)␣
↪/ 2;

        int32_t diff_y = child_y_center - cont_y_center;
        diff_y = LV_ABS(diff_y);

        /*Get the x of diff_y on a circle.*/
        int32_t x;
```

(continues on next page)

(continua dalla pagina precedente)

```
        /*If diff_y is out of the circle use the last point of the circle␣
↪(the radius)*/
        if(diff_y >= r) {
            x = r;
        }
        else {
            /*Use Pythagoras theorem to get x from radius and y*/
            uint32_t x_sqr = r * r - diff_y * diff_y;
            lv_sqrt_res_t res;
            lv_sqrt(x_sqr, &res, 0x8000);   /*Use lvgl's built in sqrt␣
↪root function*/
            x = r - res.i;
        }

        /*Translate the item by the calculated X coordinate*/
        lv_obj_set_style_translate_x(child, x, 0);

        /*Use some opacity with larger translations*/
        lv_opa_t opa = (lv_opa_t)lv_map(x, 0, r, LV_OPA_TRANSP, LV_OPA_
↪COVER);
        lv_obj_set_style_opa(child, LV_OPA_COVER - opa, 0);
    }
}

/**
 * Translate the object as they scroll
 */
void lv_example_scroll_6(void)
{
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, 200, 200);
    lv_obj_center(cont);
    lv_obj_set_flex_flow(cont, LV_FLEX_FLOW_COLUMN);
    lv_obj_add_event_cb(cont, scroll_event_cb, LV_EVENT_SCROLL, NULL);
    lv_obj_set_style_radius(cont, LV_RADIUS_CIRCLE, 0);
    lv_obj_set_style_clip_corner(cont, true, 0);
    lv_obj_set_scroll_dir(cont, LV_DIR_VER);
    lv_obj_set_scroll_snap_y(cont, LV_SCROLL_SNAP_CENTER);
    lv_obj_set_scrollbar_mode(cont, LV_SCROLLBAR_MODE_OFF);

    uint32_t i;
    for(i = 0; i < 20; i++) {
        lv_obj_t * btn = lv_button_create(cont);
        lv_obj_set_width(btn, lv_pct(100));

        lv_obj_t * label = lv_label_create(btn);
        lv_label_set_text_fmt(label, "Button %" LV_PRIu32, i);
    }

    /*Update the buttons position manually for first*/
    lv_obj_send_event(cont, LV_EVENT_SCROLL, NULL);

    /*Be sure the fist button is in the middle*/
    lv_obj_scroll_to_view(lv_obj_get_child(cont, 0), LV_ANIM_OFF);
}
```

(continues on next page)

(continua dalla pagina precedente)

```
#endif
```

Scrolling infinito

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_FLEX

static lv_obj_t * high_label;
static lv_obj_t * low_label;
static int32_t top_num;
static int32_t bottom_num;
static bool update_scroll_running = false;

static lv_obj_t * load_item(lv_obj_t * parent, int32_t num)
{
    lv_obj_t * obj = (lv_obj_t *) lv_obj_create(parent);
    lv_obj_set_size(obj, LV_PCT(100), LV_SIZE_CONTENT);
    lv_obj_t * label = lv_label_create(obj);
    lv_label_set_text_fmt(label, "%" PRId32, num);
    return obj;
}

static void update_scroll(lv_obj_t * obj)
{
    /* do not re-enter this function when `lv_obj_scroll_by`
     * triggers this callback again.
     */
    if(update_scroll_running) return;
    update_scroll_running = true;

    int32_t top_num_original = top_num;
    int32_t bottom_num_original = bottom_num;

    /* load items we're getting close to */
    while(bottom_num > -30 && lv_obj_get_scroll_bottom(obj) < 200) {
        bottom_num -= 1;
        load_item(obj, bottom_num);
        lv_obj_update_layout(obj);
        LV_LOG_USER("loaded bottom num: %" PRId32, bottom_num);
    }
    while(top_num < 30 && lv_obj_get_scroll_top(obj) < 200) {
        top_num += 1;
        int32_t bottom_before = lv_obj_get_scroll_bottom(obj);
        lv_obj_t * new_item = load_item(obj, top_num);
        lv_obj_move_to_index(new_item, 0);
        lv_obj_update_layout(obj);
        int32_t bottom_after = lv_obj_get_scroll_bottom(obj);
        lv_obj_scroll_by(obj, 0, bottom_before - bottom_after, LV_ANIM_
↪OFF);
        LV_LOG_USER("loaded top num: %" PRId32, top_num);
    }

    /* delete far-away items */
    while(lv_obj_get_scroll_bottom(obj) > 600) {
        bottom_num += 1;
        lv_obj_t * child = lv_obj_get_child(obj, -1);
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_obj_delete(child);
        lv_obj_update_layout(obj);
        LV_LOG_USER("deleted bottom num: %" PRId32, bottom_num);
    }
    while(lv_obj_get_scroll_top(obj) > 600) {
        top_num -= 1;
        int32_t bottom_before = lv_obj_get_scroll_bottom(obj);
        lv_obj_t * child = lv_obj_get_child(obj, 0);
        lv_obj_delete(child);
        lv_obj_update_layout(obj);
        int32_t bottom_after = lv_obj_get_scroll_bottom(obj);
        lv_obj_scroll_by(obj, 0, bottom_before - bottom_after, LV_ANIM_
↪OFF);
        LV_LOG_USER("deleted top num: %" PRId32, top_num);
    }

    if(top_num != top_num_original) {
        lv_label_set_text_fmt(high_label, "current largest\nloaded value:\n
↪%" PRId32, top_num);
    }
    if(bottom_num != bottom_num_original) {
        lv_label_set_text_fmt(low_label, "current smallest\nloaded value:\n
↪%" PRId32, bottom_num);
    }

    update_scroll_running = false;
}

static void scroll_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    update_scroll(obj);
}

static void checkbox_cb(lv_event_t * e)
{
    lv_obj_t * checkbox = lv_event_get_target_obj(e);
    lv_obj_t * obj = (lv_obj_t *) lv_event_get_user_data(e);
    bool checked = lv_obj_has_state(checkbox, LV_STATE_CHECKED);
    lv_obj_set_style_opa(obj, checked ? LV_OPA_COVER : LV_OPA_TRANSP, LV_
↪PART_SCROLLBAR);
}

/**
 * Dynamically load widgets while scrolling
 */
void lv_example_scroll_7(void)
{
    lv_obj_t * scr = lv_screen_active();
    lv_obj_t * obj = lv_obj_create(scr);
    lv_obj_set_size(obj, 160, 220);
    lv_obj_align(obj, LV_ALIGN_RIGHT_MID, -10, 0);
    lv_obj_set_flex_flow(obj, LV_FLEX_FLOW_COLUMN);
    lv_obj_set_style_opa(obj, LV_OPA_TRANSP, LV_PART_SCROLLBAR);

    high_label = lv_label_create(scr);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_label_set_text_static(high_label, "current largest\nloaded value:");
    lv_obj_align(high_label, LV_ALIGN_TOP_LEFT, 10, 10);

    lv_obj_t * checkbox = lv_checkbox_create(scr);
    lv_checkbox_set_text_static(checkbox, "show\nscrollbar");
    lv_obj_align(checkbox, LV_ALIGN_LEFT_MID, 10, 0);
    lv_obj_add_event_cb(checkbox, checkbox_cb, LV_EVENT_VALUE_CHANGED,␣
↪obj);

    low_label = lv_label_create(scr);
    lv_label_set_text_static(low_label, "current smallest\nloaded value:");
    lv_obj_align(low_label, LV_ALIGN_BOTTOM_LEFT, 10, -10);

    load_item(obj, 3);
    /* These counters hold the the highest/lowest number currently loaded.␣
↪*/
    top_num = 3;
    bottom_num = 3;

    lv_obj_update_layout(obj);
    update_scroll(obj);
    lv_obj_add_event_cb(obj, scroll_cb, LV_EVENT_SCROLL, NULL);
}

#endif
```

Scrolling circolare

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_FLEX


/*
 * Circular list implementation based on:
 * Reference: https://blog.csdn.net/TQW4321/article/details/145434819
 */

/* Applies to both width (for row items) and height (for column items) */
#define ITEM_SIZE     80

/* Function to calculate the total content width of the container */
static int32_t get_content_width(lv_obj_t * cont)
{
    int32_t w = 0;
    int32_t pad_column = lv_obj_get_style_pad_column(cont, LV_PART_MAIN); /
↪/ □□□
    int32_t total = (int32_t)lv_obj_get_child_count(cont);

    for(int32_t i = 0; i < total; i++) {
        w += lv_obj_get_width(lv_obj_get_child(cont, i));
        if(i < total - 1) w += pad_column;
    }
    return w + lv_obj_get_style_pad_left(cont, LV_PART_MAIN)
           + lv_obj_get_style_pad_right(cont, LV_PART_MAIN);
}
```

(continues on next page)

(continua dalla pagina precedente)

```
/* Scroll event callback for row layout */
static void cont_row_scroll_event_cb(lv_event_t * e)
{
    static bool is_adjusting = false;
    lv_obj_t * cont = lv_event_get_current_target_obj(e);

    if(!is_adjusting) {
        is_adjusting = true;
        int32_t scroll_x = lv_obj_get_scroll_x(cont);
        int32_t cont_w = lv_obj_get_width(cont);
        int32_t content_w = get_content_width(cont);

        /* Use ITEM_SIZE as horizontal item width */
        const int32_t item_width = ITEM_SIZE;

        if(scroll_x <= 0) {
            lv_obj_t * last_child = lv_obj_get_child(cont, (int32_t)(lv_
↪obj_get_child_count(cont) - 1));
            lv_obj_move_to_index(last_child, 0);
            lv_obj_scroll_to_x(cont, scroll_x + item_width, LV_ANIM_OFF);
        }
        else if(scroll_x > content_w - cont_w) {
            lv_obj_t * first_child = lv_obj_get_child(cont, 0);
            lv_obj_move_to_index(first_child, (int32_t)(lv_obj_get_child_
↪count(cont) - 1));
            lv_obj_scroll_to_x(cont, scroll_x - item_width, LV_ANIM_OFF);
        }
        is_adjusting = false;
    }
}

/* Function to calculate the total content height of the container */
static int32_t get_content_height(lv_obj_t * cont)
{
    int32_t h = 0;
    int32_t pad_row = lv_obj_get_style_pad_row(cont, LV_PART_MAIN);
    int32_t total = (int32_t)lv_obj_get_child_count(cont);

    for(int i = 0; i < total; i++) {
        h += lv_obj_get_height(lv_obj_get_child(cont, i));
        if(i < total - 1) h += pad_row;
    }
    return h + lv_obj_get_style_pad_top(cont, LV_PART_MAIN)
           + lv_obj_get_style_pad_bottom(cont, LV_PART_MAIN);
}

/* Scroll event callback for column layout */
static void cont_col_scroll_event_cb(lv_event_t * e)
{
    static bool is_adjusting = false;
    lv_obj_t * cont = lv_event_get_current_target_obj(e);

    if(!is_adjusting) {
        is_adjusting = true;
        int32_t scroll_y = lv_obj_get_scroll_y(cont);
        int32_t cont_h = lv_obj_get_height(cont);
```

(continues on next page)

(continua dalla pagina precedente)

```
        int32_t content_h = get_content_height(cont);

        /* Use ITEM_SIZE as vertical item height */
        const int32_t item_height = ITEM_SIZE;

        if(scroll_y <= 0) {
            lv_obj_t * last_child = lv_obj_get_child(cont, (int32_t)(lv_
↪obj_get_child_count(cont) - 1));
            lv_obj_move_to_index(last_child, 0);
            lv_obj_scroll_to_y(cont, scroll_y + item_height, LV_ANIM_OFF);
        }
        else if(scroll_y > content_h - cont_h) {
            lv_obj_t * first_child = lv_obj_get_child(cont, 0);
            lv_obj_move_to_index(first_child, (int32_t)(lv_obj_get_child_
↪count(cont) - 1));
            lv_obj_scroll_to_y(cont, scroll_y - item_height, LV_ANIM_OFF);
        }
        is_adjusting = false;
    }
}

void lv_example_scroll_8(void)
{
    /* Create a scroll container with ROW flex direction */
    lv_obj_t * cont_row = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont_row, 300, 75);
    lv_obj_align(cont_row, LV_ALIGN_TOP_MID, 0, 5);
    lv_obj_set_flex_flow(cont_row, LV_FLEX_FLOW_ROW);
    lv_obj_add_event_cb(cont_row, cont_row_scroll_event_cb, LV_EVENT_
↪SCROLL, NULL);
    /* Hide scrollbar visuals */
    lv_obj_set_scrollbar_mode(cont_row, LV_SCROLLBAR_MODE_OFF);

    /* Create a scroll container with COLUMN flex direction */
    lv_obj_t * cont_col = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont_col, 200, 150);
    lv_obj_align_to(cont_col, cont_row, LV_ALIGN_OUT_BOTTOM_MID, 0, 5);
    lv_obj_set_flex_flow(cont_col, LV_FLEX_FLOW_COLUMN);
    lv_obj_add_event_cb(cont_col, cont_col_scroll_event_cb, LV_EVENT_
↪SCROLL, NULL);
    /* Hide scrollbar visuals */
    lv_obj_set_scrollbar_mode(cont_col, LV_SCROLLBAR_MODE_OFF);

    /* If the number of items is less than 3, scrolling may not be needed␣
↪or may cause unexpected behavior. */
    uint32_t i;
    for(i = 0; i < 10; i++) {
        lv_obj_t * obj;
        lv_obj_t * label;

        /*Add items to the row*/
        obj = lv_button_create(cont_row);
        lv_obj_set_size(obj, ITEM_SIZE, LV_PCT(100));

        label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "Item %" LV_PRIu32, i + 1);
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_obj_center(label);

        /*Add items to the column*/
        obj = lv_button_create(cont_col);
        lv_obj_set_size(obj, LV_PCT(100), ITEM_SIZE);

        label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "Item %" LV_PRIu32, i + 1);
        lv_obj_center(label);
    }
}


#endif
```

5.7.6 API

CAPITOLO 6

All Widgets

6.1 Base Widget (lv_obj)

The most fundamental of all Widgets is the Base Widget, on which all other widgets are based. From an Object-Oriented perspective, think of the Base Widget as the Widget class from which all other Widgets inherit.

By this mechanism, all Widgets carry *the features of the Base Widget*. Therefore, the functions and functionalities of the Base Widget can be used with other widgets as well. For example `lv_obj_set_width(slider, 100)`.

The Base Widget can be used directly as a simple widget. While it is a simple rectangle, it has a large number of features shared with all Widgets, detailed in *Common Widget Features*. In HTML terms, think of it as a `<div>`.

6.2 3D Texture (lv_3dtexture)

6.2.1 Panoramica

3D texture widgets are used to embed an external 3D graphics library's "texture" primitive into an LVGL UI.

6.2.2 Parts and Styles

The size should be set to the actual size of the texture primitive. The opacity may also be changed.

6.2.3 Utilizzo

Requires a draw unit to be enabled which can draw the `LV_DRAW_TASK_TYPE_3D` draw task type.

OpenGL is the first supported 3D graphics back-end. The following must be enabled.

- `LV_USE_3DTEXTURE`
- `LV_USE_OPENGLES`
- `LV_USE_DRAW_OPENGLES`

See *LVGL's OpenGLES driver docs* to create a window and a display texture.

The `lv_example_3dtexture <https://github.com/lvgl/lv_example_3dtexture>` repository is a demonstration of how to use the `3dtexture` widget to load glTF models with OpenGL and display them in LVGL.

```
lv_obj_t * tex = lv_3dtexture_create(parent);
/* Render something to the texture. You can replace it with your code. */
lv_3dtexture_id_t gltf_texture = render_gltf_model_to_opengl_texture(path,␣
↪w, h, color);
lv_3dtexture_set_src(tex, gltf_texture);
lv_obj_set_size(tex, w, h);
lv_obj_set_style_opa(tex, opa, 0);
```

The real type of `lv_3dtexture_id_t` depends on the active 3D graphics back-end. With OpenGL, the type is `unsigned int`.

6.2.4 Eventi

3D Texture has no special event handling.

6.2.5 Esempio

See the `lv_example_3dtexture <https://github.com/lvgl/lv_example_3dtexture>` repo.

6.2.6 API

lv_3dtexture.h

lv_3dtexture_private.h

6.3 Animation Image (lv_animimg)

6.3.1 Panoramica

The animation image is similar to the normal 'Image' Widget. The only difference is that instead of one source image, you set an array of multiple source images that supply "frames" in an animation.

You can specify a duration and repeat count.

6.3.2 Parts and Styles

- *`LV_PART_MAIN`* A background rectangle that uses the *typical background style properties* and the image itself using the image style properties.

6.3.3 Utilizzo

Image sources

To set the image animation image sources, use *`lv_animimg_set_src`*`(animimg, dsc[], num)`.

To set the images source for flip playback of animation image, use *`lv_animimg_set_src_reverse`*`(animimg, dsc[], num)`.

Using the inner animation

For more advanced use cases, the animation used internally by the image can be retrieved using *`lv_animimg_get_anim`*`(animimg)`. Using this, you can use the *Animation* functions, for example, to override the animation values using the *`lv_anim_set_values`*`(anim, start, end)` or to set a callback on the animation completed event.

6.3.4 Eventi

No special events are sent by Animation-Image Widgets.

Further Reading

Learn more about *Base-Widget Events* emitted by all Widgets.

Learn more about *Eventi*.

6.3.5 Keys

No *Keys* are processed by Animation-Image Widgets.

Further Reading

Learn more about *Keys*.

6.3.6 Esempio

Immagine animata semplice

```
#include "../../lv_examples.h"
#if LV_USE_ANIMIMG && LV_BUILD_EXAMPLES
LV_IMAGE_DECLARE(animimg001);
LV_IMAGE_DECLARE(animimg002);
LV_IMAGE_DECLARE(animimg003);

static const lv_image_dsc_t * anim_imgs[3] = {
    &animimg001,
    & animimg002,
    & animimg003,
};

void lv_example_animimg_1(void)
{
    lv_obj_t * animimg0 = lv_animimg_create(lv_screen_active());
    lv_obj_center(animimg0);
    lv_animimg_set_src(animimg0, (const void **) anim_imgs, 3);
    lv_animimg_set_duration(animimg0, 1000);
    lv_animimg_set_repeat_count(animimg0, LV_ANIM_REPEAT_INFINITE);
    lv_animimg_start(animimg0);
}

#endif
```

6.3.7 API

lv_animimage.h

lv_animimage_private.h

6.4 Arc (lv_arc)

6.4.1 Panoramica

The Arc consists of a background and a foreground arc. The foreground (indicator) can be touch-adjusted.

6.4.2 Parts and Styles

- `LV_PART_MAIN` Draws a background using the typical background style properties and an arc using the Arc style properties. The Arc's size and position will respect the *padding* style properties.
- `LV_PART_INDICATOR` Draws another Arc using the *Arc* style properties. Its padding values are interpreted relative to the background Arc.
- `LV_PART_KNOB` Draws a handle on the end of the indicator using all background properties and padding values. With zero padding the knob size is the same as the indicator's width. Larger padding makes it larger, smaller padding makes it smaller.

6.4.3 Utilizzo

Value and range

A new value can be set using `lv_arc_set_value(arc, new_value)`. The value is interpreted in a range (minimum and maximum values) which can be modified with `lv_arc_set_range(arc, min, max)`. The default range is 0..100.

The indicator Arc is drawn on the main part's Arc. Thus, if the value is set to maximum, the indicator Arc will cover the entire "background" Arc. To set the start and end angle of the background Arc use any of these functions:

- `lv_arc_set_bg_start_angle(arc, angle)`
- `lv_arc_set_bg_end_angle(arc, angle)`
- `lv_arc_set_bg_angles(arc, start_angle, end_angle)`

Zero degrees is at the middle right (3 o'clock) of the Widget and the degrees increasing in the clockwise direction. The angle values should be in the range [0..360].

Rotation

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

Mode

The Arc can be one of the following modes:

- `LV_ARC_MODE_NORMAL` Indicator Arc is drawn clockwise from minimum to current value.
- `LV_ARC_MODE_REVERSE` Indicator Arc is drawn counter-clockwise from maximum to current value.
- `LV_ARC_MODE_SYMMETRICAL` Indicator Arc is drawn from middle point to current value.

The mode can be set by `lv_arc_set_mode(arc, LV_ARC_MODE_...)` and has no effect until angle is set by `lv_arc_set_value()` or value of the Arc is changed by pointer input (finger, mouse, etc.).

Change rate

When the Arc's value is changed by pointer input (finger, mouse, etc.), the rate of its change is limited according to its *change rate*. Change rate is defined in degrees/second units and can be set with `lv_arc_set_change_rate(arc, rate)`

Knob offset

Changing the knob offset allows the location of the knob to be moved relative to the end of the Arc. The knob offset can be set by `lv_arc_set_knob_offset(arc, offset_angle)`, and will only be visible if `LV_PART_KNOB` is visible.

Setting indicator programmatically

It is possible to set indicator angle directly with any of these functions:

- `lv_arc_set_start_angle(arc, start_angle)`
- `lv_arc_set_end_angle(arc, end_angle)`
- `lv_arc_set_angles(arc, start_angle, end_angle)`

When used, "value" and "mode" are ignored.

In other words, the angle and value settings are independent. You should exclusively use one or the other of the two methods. Mixing the two could result in unintended behavior.

To make the arc non-adjustable, remove the style of the knob and make the Widget non-clickable:

```
lv_obj_remove_style(arc, NULL, LV_PART_KNOB);
lv_obj_remove_flag(arc, LV_OBJ_FLAG_CLICKABLE);
```

Interactive area

By default `LV_OBJ_FLAG_ADV_HITTEST` is disabled which means the Arc's whole area is interactive. As usual `lv_obj_set_ext_click_area()` can be used to increase the area that will respond to pointer input (touch, mouse, etc.) outside the Arc by a specified number of pixels.

If `LV_OBJ_FLAG_ADV_HITTEST` is enabled the Arc will be sensitive only in the range between start and end background angles and on the Arc itself (not inside the Arc). In this case `ext_click_area` makes the sensitive area ticker both inward and outward. Additionally, a tolerance of `lv_dpx(50)` pixels is applied to each angle, extending the hit-test range along the Arc's length.

Place another Widget on the knob

Another Widget can be positioned according to the current position of the Arc in order to follow the Arc's current value (angle). To do this use `lv_arc_align_obj_to_angle(arc, widget_to_align, radius_offset)`.

Similarly `lv_arc_rotate_obj_to_angle(arc, widget_to_rotate, radius_offset)` can be used to rotate the Widget to the current value of the Arc.

A typical use case is to call these functions in the `VALUE_CHANGED` event of the Arc.

6.4.4 Eventi

- `LV_EVENT_VALUE_CHANGED` sent when Arc is pressed/dragged to a new value.

> **Further Reading**
>
> Learn more about *Base-Widget Events* emitted by all Widgets.
>
> Learn more about *Eventi*.

6.4.5 Keys

- `LV_KEY_RIGHT/UP` Increases value by one.
- `LV_KEY_LEFT/DOWN` Decreases value by one.

Further Reading

Learn more about *Keys*.

6.4.6 Esempio

Arco semplice

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

#if LV_USE_ARC && LV_BUILD_EXAMPLES

static void value_changed_event_cb(lv_event_t * e);

void lv_example_arc_1(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());

    /*Create an Arc*/
    lv_obj_t * arc = lv_arc_create(lv_screen_active());
    lv_obj_set_size(arc, 150, 150);
    lv_arc_set_rotation(arc, 135);
    lv_arc_set_bg_angles(arc, 0, 270);
    lv_arc_set_value(arc, 10);
    lv_obj_center(arc);
    lv_obj_add_event_cb(arc, value_changed_event_cb, LV_EVENT_VALUE_
↪CHANGED, label);

    /*Manually update the label for the first time*/
    lv_obj_send_event(arc, LV_EVENT_VALUE_CHANGED, NULL);
}

static void value_changed_event_cb(lv_event_t * e)
{
    lv_obj_t * arc = lv_event_get_target_obj(e);
    lv_obj_t * label = (lv_obj_t *)lv_event_get_user_data(e);

    lv_label_set_text_fmt(label, "%" LV_PRId32 "%%", lv_arc_get_
↪value(arc));

    /*Rotate the label to the current position of the arc*/
    lv_arc_rotate_obj_to_angle(arc, label, 25);
}

#endif
```

Loader con Arc

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

#if LV_USE_ARC && LV_BUILD_EXAMPLES

static void set_angle(void * obj, int32_t v)
{
    lv_arc_set_value((lv_obj_t *)obj, v);
}
```

(continues on next page)

(continua dalla pagina precedente)

```
/**
 * Create an arc which acts as a loader.
 */
void lv_example_arc_2(void)
{
    /*Create an Arc*/
    lv_obj_t * arc = lv_arc_create(lv_screen_active());
    lv_arc_set_rotation(arc, 270);
    lv_arc_set_bg_angles(arc, 0, 360);
    lv_obj_remove_style(arc, NULL, LV_PART_KNOB);   /*Be sure the knob is␣
↪not displayed*/
    lv_obj_remove_flag(arc, LV_OBJ_FLAG_CLICKABLE);  /*To not allow␣
↪adjusting by click*/
    lv_obj_center(arc);

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, arc);
    lv_anim_set_exec_cb(&a, set_angle);
    lv_anim_set_duration(&a, 1000);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);    /*Just for␣
↪the demo*/
    lv_anim_set_repeat_delay(&a, 500);
    lv_anim_set_values(&a, 0, 100);
    lv_anim_start(&a);

}

#endif
```

Grafico a torta con sezioni cliccabili tramite Archi

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

#if LV_USE_ARC && LV_BUILD_EXAMPLES

#define CHART_SIZE 150
#define SLICE_OFFSET 20

typedef struct {
    int start_angle;
    int end_angle;
    int mid_angle;
    lv_point_t home;
    bool out;
} slice_info_t;

typedef struct {
    lv_obj_t * obj;
    int start_x;
    int start_y;
    int end_x;
    int end_y;
} slice_anim_data_t;
```

(continues on next page)

(continua dalla pagina precedente)

```
static float angle_accum = 0.0f;
static slice_info_t * active_info = NULL;
static lv_obj_t * active_arc = NULL;

static void anim_move_cb(void * var, int32_t v)
{
    slice_anim_data_t * d = (slice_anim_data_t *) var;

    int32_t x = d->start_x + ((d->end_x - d->start_x) * v) / 100;
    int32_t y = d->start_y + ((d->end_y - d->start_y) * v) / 100;
    lv_obj_set_pos(d->obj, x, y);
}

static void anim_cleanup_cb(lv_anim_t * a)
{
    lv_free(a->var);
}

static void arc_click_cb(lv_event_t * e)
{
    lv_obj_t * arc  = lv_event_get_target_obj(e);
    slice_info_t * info = (slice_info_t *)lv_event_get_user_data(e);

    int32_t x_off = (SLICE_OFFSET * lv_trigo_cos(info->mid_angle)) >> LV_
↪TRIGO_SHIFT;
    int32_t y_off = (SLICE_OFFSET * lv_trigo_sin(info->mid_angle)) >> LV_
↪TRIGO_SHIFT;

    if(active_info && active_info != info && active_info->out) {
        slice_anim_data_t * anim_back = (slice_anim_data_t *) lv_
↪malloc(sizeof(slice_anim_data_t));
        anim_back->obj = active_arc;
        anim_back->start_x = lv_obj_get_x(active_arc) - SLICE_OFFSET;
        anim_back->start_y = lv_obj_get_y(active_arc) - SLICE_OFFSET;
        anim_back->end_x = active_info->home.x;
        anim_back->end_y = active_info->home.y;

        active_info->out = false;

        lv_anim_t a;
        lv_anim_init(&a);
        lv_anim_set_var(&a, anim_back);
        lv_anim_set_exec_cb(&a, anim_move_cb);
        lv_anim_set_time(&a, 200);
        lv_anim_set_values(&a, 0, 100);
        lv_anim_set_deleted_cb(&a, anim_cleanup_cb);
        lv_anim_start(&a);
    }

    int target_x, target_y;
    if(info->out) {
        target_x  = info->home.x;
        target_y  = info->home.y;
        info->out = false;
        active_info = NULL;
        active_arc = NULL;
```

(continues on next page)

(continua dalla pagina precedente)

```
    }
    else {
        target_x  = info->home.x + x_off;
        target_y  = info->home.y + y_off;
        info->out = true;
        active_info = info;
        active_arc = arc;
    }

    slice_anim_data_t * anim_data = (slice_anim_data_t *) lv_
↪malloc(sizeof(slice_anim_data_t));
    anim_data->obj     = arc;
    anim_data->start_x = lv_obj_get_x(arc) - SLICE_OFFSET;
    anim_data->start_y = lv_obj_get_y(arc) - SLICE_OFFSET;
    anim_data->end_x   = target_x;
    anim_data->end_y   = target_y;

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, anim_data);
    lv_anim_set_exec_cb(&a, anim_move_cb);
    lv_anim_set_time(&a, 200);
    lv_anim_set_values(&a, 0, 100);
    lv_anim_set_deleted_cb(&a, anim_cleanup_cb);
    lv_anim_start(&a);
}

static void create_slice(lv_obj_t * parent, int percentage, lv_color_t␣
↪color)
{
    if(percentage <= 0) return;

    float slice_angle = (percentage * 360.0f) / 100.0f;
    int start = (int)(angle_accum + 0.5f);
    angle_accum += slice_angle;
    int end = (int)(angle_accum + 0.5f);
    if(end > 360) end = 360;

    lv_obj_t * arc = lv_arc_create(parent);
    lv_obj_set_size(arc, CHART_SIZE, CHART_SIZE);
    lv_obj_center(arc);

    lv_arc_set_mode(arc, LV_ARC_MODE_NORMAL);
    lv_arc_set_bg_start_angle(arc, start);
    lv_arc_set_bg_end_angle(arc, end);

    lv_obj_set_style_arc_width(arc, CHART_SIZE / 2, LV_PART_MAIN);
    lv_obj_set_style_arc_width(arc, 0, LV_PART_INDICATOR);

    lv_obj_set_style_arc_color(arc, color, LV_PART_MAIN);
    lv_obj_set_style_arc_rounded(arc, false, LV_PART_MAIN);
    lv_obj_remove_style(arc, NULL, LV_PART_KNOB);
    lv_obj_add_flag(arc, LV_OBJ_FLAG_ADV_HITTEST);

    lv_obj_t * label = lv_label_create(arc);
    lv_label_set_text_fmt(label, "%d%%", percentage);
```

(continues on next page)

(continua dalla pagina precedente)

```
    int mid_angle = start + ((end - start) / 2);
    int radius = CHART_SIZE / 4;
    int x_offset = (radius * lv_trigo_cos(mid_angle)) >> LV_TRIGO_SHIFT;
    int y_offset = (radius * lv_trigo_sin(mid_angle)) >> LV_TRIGO_SHIFT;

    lv_obj_align(label, LV_ALIGN_CENTER, x_offset, y_offset);

    slice_info_t * info = (slice_info_t *) lv_malloc(sizeof(slice_info_t));
    info->start_angle = start;
    info->end_angle = end;
    info->mid_angle = mid_angle;
    info->out         = false;
    info->home.x = lv_obj_get_x(arc);
    info->home.y = lv_obj_get_y(arc);
    lv_obj_add_event_cb(arc, arc_click_cb, LV_EVENT_CLICKED, info);
}

void lv_example_arc_3(void)
{
    /* Root container: flex row */
    lv_obj_t * root = lv_obj_create(lv_screen_active());
    lv_obj_set_size(root, LV_SIZE_CONTENT, LV_SIZE_CONTENT);
    lv_obj_center(root);
    lv_obj_set_flex_flow(root, LV_FLEX_FLOW_ROW);
    lv_obj_set_flex_align(root, LV_FLEX_ALIGN_CENTER, LV_FLEX_ALIGN_CENTER,
↪ LV_FLEX_ALIGN_CENTER);

    lv_obj_set_style_pad_all(root, 0, LV_PART_MAIN);
    lv_obj_set_style_border_width(root, 0, LV_PART_MAIN);
    lv_obj_set_style_border_color(root, lv_palette_main(LV_PALETTE_RED),␣
↪LV_PART_MAIN);
    lv_obj_set_style_bg_opa(root, LV_OPA_TRANSP, LV_PART_MAIN);
    lv_obj_remove_flag(root, LV_OBJ_FLAG_SCROLLABLE);

    /* Slices container */
    lv_obj_t * slices_container = lv_obj_create(root);
    lv_obj_set_size(slices_container, CHART_SIZE + 2 * SLICE_OFFSET, CHART_
↪SIZE + 2 * SLICE_OFFSET);
    lv_obj_set_style_pad_all(slices_container, 0, LV_PART_MAIN);
    lv_obj_set_style_margin_all(slices_container, 0, LV_PART_MAIN);
    lv_obj_set_style_border_width(slices_container, 0, LV_PART_MAIN);
    lv_obj_set_style_border_color(slices_container, lv_palette_main(LV_
↪PALETTE_BLUE), LV_PART_MAIN);
    lv_obj_set_style_bg_opa(slices_container, LV_OPA_TRANSP, LV_PART_MAIN);
    lv_obj_remove_flag(slices_container, LV_OBJ_FLAG_SCROLLABLE);

    /* Create slices */
    angle_accum = 0.0f;
    create_slice(slices_container, 12, lv_palette_main(LV_PALETTE_RED));
    create_slice(slices_container, 18, lv_palette_main(LV_PALETTE_BLUE));
    create_slice(slices_container, 26, lv_palette_main(LV_PALETTE_GREEN));
    create_slice(slices_container, 24, lv_palette_main(LV_PALETTE_ORANGE));
    create_slice(slices_container, 20, lv_palette_main(LV_PALETTE_BLUE_
↪GREY));
}
```

(continues on next page)

(continua dalla pagina precedente)

```
#endif
```

6.4.7 API

lv_arc.h

lv_arc_private.h

lv_arclabel.h

lv_arclabel_private.h

lv_observer.h

6.5 Arc Label (lv_arclabel)

6.5.1 Panoramica

The Arc Label is a specialized widget designed to display text along an arc. It allows for flexible text placement and styling, making it suitable for applications where text needs to follow a curved path, such as in gauges, dials, or custom interfaces. The widget supports various configurations, including text alignment, direction, radius adjustment, and color customization.

6.5.2 Parts and Styles

- *LV_PART_MAIN* Represents the main part of the Arc Label, including the arc path and the text rendered along it. The appearance of the text and the arc can be customized using typical text and background style properties.

6.5.3 Utilizzo

Text Management

- Text can be set using *lv_arclabel_set_text*(arclabel, "Your text").
- For formatted text, use *lv_arclabel_set_text_fmt*(arclabel, "Formatted %s", "text").
- Static text can be set with *lv_arclabel_set_text_static*(arclabel, static_text), which avoids dynamic memory allocation.

Angle Configuration

- Set the starting angle of the arc with *lv_arclabel_set_angle_start*(arclabel, angle).
- Define the arc size (angular span) using *lv_arclabel_set_angle_size*(arclabel, size).
- Angles are measured in degrees, starting from the positive x-axis (3 o'clock position) and increasing clockwise.

Direction

Specify the text direction along the arc using *lv_arclabel_set_dir*(arclabel, *LV_ARCLABEL_DIR_CLOCKWISE*) or *lv_arclabel_set_dir*(arclabel, LV_ARCLABEL_DIR_COUNTERCLOCKWISE).

- *LV_ARCLABEL_DIR_CLOCKWISE* Text flows in a clockwise direction along the arc.
- LV_ARCLABEL_DIR_COUNTERCLOCKWISE Text flows in a counter-clockwise direction along the arc.

Alignment

Adjust vertical text alignment with `lv_arclabel_set_text_vertical_align(arclabel, LV_ARCLABEL_TEXT_ALIGN_CENTER)`. Set horizontal alignment using `lv_arclabel_set_text_horizontal_align(arclabel, LV_ARCLABEL_TEXT_ALIGN_CENTER)`.

Both vertical and horizontal use the same logic.

- `LV_ARCLABEL_TEXT_ALIGN_DEFAULT` Uses the default alignment.
- `LV_ARCLABEL_TEXT_ALIGN_LEADING` Aligns text to the leading edge of the arc.
- `LV_ARCLABEL_TEXT_ALIGN_CENTER` Centers text along the arc.
- `LV_ARCLABEL_TEXT_ALIGN_TRAILING` Aligns text to the trailing edge of the arc.

Radius and Center Offset

- Set the radius of the arc with `lv_arclabel_set_radius(arclabel, radius)`.
- Adjust the center offset of the arc using `lv_arclabel_set_center_offset_x(arclabel, x)` and `lv_arclabel_set_center_offset_y(arclabel, y)`.

Color and Recoloring

Enable text recoloring with `lv_arclabel_set_recolor(arclabel, true)`. This allows parts of the text to be colored differently using color commands embedded in the text string.

Interactive Behavior

By default, Arc Label is not clickable. To make it interactive, you would need to add custom event handling, as it does not inherit clickability by default.

6.5.4 Eventi

The Arc Label primarily inherits events from the base object class. It does not define specific events beyond those common to all widgets. You can attach custom event handlers to respond to interactions if needed.

6.5.5 Keys

The Arc Label does not define specific key bindings beyond those inherited from the base object class. Keyboard navigation and interaction would require additional implementation.

6.5.6 Esempio

Simple Arc Label

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

#if LV_USE_ARCLABEL && LV_BUILD_EXAMPLES

static const char * ARCLABEL_TEXT =
    "I'm on an #FA7C45 ARC#! Centered with #12c2E9 C##8B68E8 O##c471ed L#
↪#B654E5 O##C84AB2 R##DB417A F##f64659 U##ff8888 L# feature!\n";

void lv_example_arclabel_1(void)
{
    lv_obj_t * arclabel_inner = NULL;
    lv_obj_t * arclabel_outer = NULL;
    lv_obj_t * arclabel_slogan_1 = NULL;
    lv_obj_t * arclabel_slogan_2 = NULL;
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_style_bg_color(lv_screen_active(), lv_color_black(), LV_
↪PART_MAIN);

    arclabel_inner = lv_arclabel_create(lv_screen_active());
    lv_obj_set_size(arclabel_inner, 200, 200);
    lv_obj_set_style_text_color(arclabel_inner, lv_color_white(), LV_PART_
↪MAIN);
    lv_arclabel_set_text_static(arclabel_inner, ARCLABEL_TEXT);
    lv_arclabel_set_angle_start(arclabel_inner, 180);
    lv_arclabel_set_radius(arclabel_inner, LV_PCT(80));
    lv_arclabel_set_recolor(arclabel_inner, true);
    lv_arclabel_set_text_vertical_align(arclabel_inner, LV_ARCLABEL_TEXT_
↪ALIGN_TRAILING);
    lv_arclabel_set_dir(arclabel_inner, LV_ARCLABEL_DIR_COUNTER_CLOCKWISE);
    lv_arclabel_set_text_horizontal_align(arclabel_inner, LV_ARCLABEL_TEXT_
↪ALIGN_CENTER);
    lv_obj_center(arclabel_inner);

    arclabel_outer = lv_arclabel_create(lv_screen_active());
    lv_obj_set_size(arclabel_outer, 200, 200);
    lv_obj_set_style_text_letter_space(arclabel_outer, 2, LV_PART_MAIN);
    lv_obj_set_style_text_color(arclabel_outer, lv_color_hex(0x888888), LV_
↪PART_MAIN);
    lv_arclabel_set_angle_start(arclabel_outer, -180);
    lv_arclabel_set_text_static(arclabel_outer, ARCLABEL_TEXT);
    lv_arclabel_set_radius(arclabel_outer, LV_PCT(100));
    lv_arclabel_set_recolor(arclabel_outer, true);
    lv_arclabel_set_text_vertical_align(arclabel_outer, LV_ARCLABEL_TEXT_
↪ALIGN_LEADING);
    lv_arclabel_set_dir(arclabel_outer, LV_ARCLABEL_DIR_CLOCKWISE);
    lv_arclabel_set_text_horizontal_align(arclabel_outer, LV_ARCLABEL_TEXT_
↪ALIGN_CENTER);
    lv_obj_center(arclabel_outer);

    arclabel_slogan_1 = lv_arclabel_create(lv_screen_active());
    lv_obj_set_size(arclabel_slogan_1, 300, 200);
    lv_obj_set_style_text_letter_space(arclabel_slogan_1, 2, LV_PART_MAIN);
    lv_obj_set_style_text_color(arclabel_slogan_1, lv_palette_main(LV_
↪PALETTE_AMBER), LV_PART_MAIN);
    lv_arclabel_set_text_static(arclabel_slogan_1, "STAY HUNGRY");
    lv_arclabel_set_offset(arclabel_slogan_1, 30);
    lv_arclabel_set_radius(arclabel_slogan_1, 150);
    lv_arclabel_set_recolor(arclabel_slogan_1, true);
    lv_arclabel_set_text_vertical_align(arclabel_slogan_1, LV_ARCLABEL_
↪TEXT_ALIGN_TRAILING);
    lv_arclabel_set_text_horizontal_align(arclabel_slogan_1, LV_ARCLABEL_
↪TEXT_ALIGN_CENTER);
    lv_arclabel_set_dir(arclabel_slogan_1, LV_ARCLABEL_DIR_COUNTER_
↪CLOCKWISE);
    lv_obj_center(arclabel_slogan_1);

    arclabel_slogan_2 = lv_arclabel_create(lv_screen_active());
    lv_obj_set_size(arclabel_slogan_2, 300, 200);
    lv_obj_set_style_text_letter_space(arclabel_slogan_2, 2, LV_PART_MAIN);
    lv_obj_set_style_text_color(arclabel_slogan_2, lv_palette_main(LV_
↪PALETTE_AMBER), LV_PART_MAIN);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_arclabel_set_text_static(arclabel_slogan_2, "STAY FOOLISH");
    lv_arclabel_set_offset(arclabel_slogan_2, 30);
    lv_arclabel_set_radius(arclabel_slogan_2, 150);
    lv_arclabel_set_angle_start(arclabel_slogan_2, 180);
    lv_arclabel_set_recolor(arclabel_slogan_2, true);
    lv_arclabel_set_text_vertical_align(arclabel_slogan_2, LV_ARCLABEL_
↪TEXT_ALIGN_TRAILING);
    lv_arclabel_set_text_horizontal_align(arclabel_slogan_2, LV_ARCLABEL_
↪TEXT_ALIGN_CENTER);
    lv_arclabel_set_dir(arclabel_slogan_2, LV_ARCLABEL_DIR_COUNTER_
↪CLOCKWISE);
    lv_obj_center(arclabel_slogan_2);


#if LV_FONT_MONTSERRAT_18
    lv_obj_set_style_text_font(arclabel_inner, &lv_font_montserrat_18, LV_
↪PART_MAIN);
    lv_obj_set_style_text_font(arclabel_outer, &lv_font_montserrat_18, LV_
↪PART_MAIN);
#endif
#if LV_FONT_MONTSERRAT_24
    lv_obj_set_style_text_font(arclabel_slogan_1, &lv_font_montserrat_24,␣
↪LV_PART_MAIN);
    lv_obj_set_style_text_font(arclabel_slogan_2, &lv_font_montserrat_24,␣
↪LV_PART_MAIN);
#endif
}

#endif
```

6.5.7 API

lv_arclabel.h

lv_arclabel_private.h

6.6 Bar (lv_bar)

6.6.1 Panoramica

The Bar Widget has a background and an indicator. The length of the indicator against the background indicates the Bar's current value.

Both the start and end values of the Bar can be set. Changing the start value to a value other than the minimum value in its range adjusts the start position of the indicator.

6.6.2 Parts and Styles

- `LV_PART_MAIN` The Bar's background. It uses the *typical background style properties*. Adding padding makes the indicator smaller or larger. The `anim_time` style property sets the animation time if the values set with `LV_ANIM_ON`.
- `LV_PART_INDICATOR` The Bar's indicator; also uses the *typical background style properties*.

6.6.3 Utilizzo

Orientation and size

- for orientation, width and height, simply set width and height style properties;
- `lv_bar_set_orientation(bar, orientation)` to override orientation caused by `width` and `height`. Valid values for `orientation` are:
  - *LV_BAR_ORIENTATION_AUTO*
  - *LV_BAR_ORIENTATION_HORIZONTAL*
  - *LV_BAR_ORIENTATION_VERTICAL*

Value and range

A new value can be set with `lv_bar_set_value(bar, new_value, LV_ANIM_ON / OFF)`. The value is interpreted in a range (minimum and maximum values) which can be modified with `lv_bar_set_range(bar, min, max)`. The default range is 0..100, and the default drawing direction is from left to right in horizontal mode and bottom to top in vertical mode. If the minimum value is greater than the maximum value, like 100..0, the drawing direction is reversed.

The new value in `lv_bar_set_value()` can be set with or without an animation depending on the last parameter (`LV_ANIM_ON/OFF`).

Modes

The Bar can be one of the following modes:

- *LV_BAR_MODE_NORMAL* A normal Bar as described above
- *LV_BAR_MODE_SYMMETRICAL* Draws indicator from zero value to current value. Requires a negative minimum value and positive maximum value, e.g. [-100..100].
- *LV_BAR_MODE_RANGE* Allows setting the start value as well with `lv_bar_set_start_value(bar, new_value, LV_ANIM_ON / OFF)`. The start value must be smaller than the end value.

6.6.4 Eventi

No special events are sent by Bar Widgets.

Further Reading

Learn more about *Base-Widget Events* emitted by all Widgets.

Learn more about *Eventi*.

6.6.5 Keys

No *Keys* are processed by Bar Widgets.

Further Reading

Learn more about *Keys*.

6.6.6 Esempio

Barra semplice

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES

void lv_example_bar_1(void)
{
    lv_obj_t * bar1 = lv_bar_create(lv_screen_active());
    lv_obj_set_size(bar1, 200, 20);
    lv_obj_center(bar1);
    lv_bar_set_value(bar1, 70, LV_ANIM_OFF);
}

#endif
```

Stile di una barra

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES

/**
 * Example of styling the bar
 */
void lv_example_bar_2(void)
{
    static lv_style_t style_bg;
    static lv_style_t style_indic;

    lv_style_init(&style_bg);
    lv_style_set_border_color(&style_bg, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_border_width(&style_bg, 2);
    lv_style_set_pad_all(&style_bg, 6); /*To make the indicator smaller*/
    lv_style_set_radius(&style_bg, 6);
    lv_style_set_anim_duration(&style_bg, 1000);

    lv_style_init(&style_indic);
    lv_style_set_bg_opa(&style_indic, LV_OPA_COVER);
    lv_style_set_bg_color(&style_indic, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_radius(&style_indic, 3);

    lv_obj_t * bar = lv_bar_create(lv_screen_active());
    lv_obj_remove_style_all(bar);  /*To have a clean start*/
    lv_obj_add_style(bar, &style_bg, 0);
    lv_obj_add_style(bar, &style_indic, LV_PART_INDICATOR);

    lv_obj_set_size(bar, 200, 20);
    lv_obj_center(bar);
    lv_bar_set_value(bar, 100, LV_ANIM_ON);
}

#endif
```

Indicatore di temperatura

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES

static void set_temp(void * bar, int32_t temp)
```

(continues on next page)

(continua dalla pagina precedente)

```
{
    lv_bar_set_value((lv_obj_t *)bar, temp, LV_ANIM_ON);
}

/**
 * A temperature meter example
 */
void lv_example_bar_3(void)
{
    static lv_style_t style_indic;

    lv_style_init(&style_indic);
    lv_style_set_bg_opa(&style_indic, LV_OPA_COVER);
    lv_style_set_bg_color(&style_indic, lv_palette_main(LV_PALETTE_RED));
    lv_style_set_bg_grad_color(&style_indic, lv_palette_main(LV_PALETTE_
↪BLUE));
    lv_style_set_bg_grad_dir(&style_indic, LV_GRAD_DIR_VER);

    lv_obj_t * bar = lv_bar_create(lv_screen_active());
    lv_obj_add_style(bar, &style_indic, LV_PART_INDICATOR);
    lv_obj_set_size(bar, 20, 200);
    lv_obj_center(bar);
    lv_bar_set_range(bar, -20, 40);

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_exec_cb(&a, set_temp);
    lv_anim_set_duration(&a, 3000);
    lv_anim_set_reverse_duration(&a, 3000);
    lv_anim_set_var(&a, bar);
    lv_anim_set_values(&a, -20, 40);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);
    lv_anim_start(&a);
}

#endif
```

Motivo a strisce e valore dell'intervallo

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES

/**
 * Bar with stripe pattern and ranged value
 */
void lv_example_bar_4(void)
{
    LV_IMAGE_DECLARE(img_skew_strip);
    static lv_style_t style_indic;

    lv_style_init(&style_indic);
    lv_style_set_bg_image_src(&style_indic, &img_skew_strip);
    lv_style_set_bg_image_tiled(&style_indic, true);
    lv_style_set_bg_image_opa(&style_indic, LV_OPA_30);

    lv_obj_t * bar = lv_bar_create(lv_screen_active());
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_add_style(bar, &style_indic, LV_PART_INDICATOR);

    lv_obj_set_size(bar, 260, 20);
    lv_obj_center(bar);
    lv_bar_set_mode(bar, LV_BAR_MODE_RANGE);
    lv_bar_set_value(bar, 90, LV_ANIM_OFF);
    lv_bar_set_start_value(bar, 20, LV_ANIM_OFF);
}

#endif
```

Barra con direzione di base LTR e RTL

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES

/**
 * Bar with LTR and RTL base direction
 */
void lv_example_bar_5(void)
{
    lv_obj_t * label;

    lv_obj_t * bar_ltr = lv_bar_create(lv_screen_active());
    lv_obj_set_size(bar_ltr, 200, 20);
    lv_bar_set_value(bar_ltr, 70, LV_ANIM_OFF);
    lv_obj_align(bar_ltr, LV_ALIGN_CENTER, 0, -30);

    label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Left to Right base direction");
    lv_obj_align_to(label, bar_ltr, LV_ALIGN_OUT_TOP_MID, 0, -5);

    lv_obj_t * bar_rtl = lv_bar_create(lv_screen_active());
    lv_obj_set_style_base_dir(bar_rtl, LV_BASE_DIR_RTL, 0);
    lv_obj_set_size(bar_rtl, 200, 20);
    lv_bar_set_value(bar_rtl, 70, LV_ANIM_OFF);
    lv_obj_align(bar_rtl, LV_ALIGN_CENTER, 0, 30);

    label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Right to Left base direction");
    lv_obj_align_to(label, bar_rtl, LV_ALIGN_OUT_TOP_MID, 0, -5);
}

#endif
```

Cassetto personalizzato per visualizzare il valore corrente

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES

#define MAX_VALUE 100
#define MIN_VALUE 0

static void set_value(void * bar, int32_t v)
{
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_bar_set_value((lv_obj_t *)bar, v, LV_ANIM_OFF);
}

static void event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);

    lv_draw_label_dsc_t label_dsc;
    lv_draw_label_dsc_init(&label_dsc);
    label_dsc.font = LV_FONT_DEFAULT;

    char buf[8];
    lv_snprintf(buf, sizeof(buf), "%d", (int)lv_bar_get_value(obj));

    lv_point_t txt_size;
    lv_text_get_size(&txt_size, buf, label_dsc.font, label_dsc.letter_
↪space, label_dsc.line_space, LV_COORD_MAX,
                     label_dsc.flag);

    lv_area_t txt_area;
    txt_area.x1 = 0;
    txt_area.x2 = txt_size.x - 1;
    txt_area.y1 = 0;
    txt_area.y2 = txt_size.y - 1;

    lv_area_t indic_area;
    lv_obj_get_coords(obj, &indic_area);
    lv_area_set_width(&indic_area, lv_area_get_width(&indic_area) * lv_bar_
↪get_value(obj) / MAX_VALUE);

    /*If the indicator is long enough put the text inside on the right*/
    if(lv_area_get_width(&indic_area) > txt_size.x + 20) {
        lv_area_align(&indic_area, &txt_area, LV_ALIGN_RIGHT_MID, -10, 0);
        label_dsc.color = lv_color_white();
    }
    /*If the indicator is still short put the text out of it on the right*/
    else {
        lv_area_align(&indic_area, &txt_area, LV_ALIGN_OUT_RIGHT_MID, 10,␣
↪0);
        label_dsc.color = lv_color_black();
    }
    label_dsc.text = buf;
    label_dsc.text_local = true;
    lv_layer_t * layer = lv_event_get_layer(e);
    lv_draw_label(layer, &label_dsc, &txt_area);
}

/**
 * Custom drawer on the bar to display the current value
 */
void lv_example_bar_6(void)
{
    lv_obj_t * bar = lv_bar_create(lv_screen_active());
    lv_bar_set_range(bar, MIN_VALUE, MAX_VALUE);
    lv_obj_set_size(bar, 200, 20);
    lv_obj_center(bar);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_add_event_cb(bar, event_cb, LV_EVENT_DRAW_MAIN_END, NULL);

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, bar);
    lv_anim_set_values(&a, 0, 100);
    lv_anim_set_exec_cb(&a, set_value);
    lv_anim_set_duration(&a, 4000);
    lv_anim_set_reverse_duration(&a, 4000);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);
    lv_anim_start(&a);

}

#endif
```

Barra con direzione opposta

```
#include "../../lv_examples.h"
#if LV_USE_BAR && LV_BUILD_EXAMPLES

/**
 * Bar with opposite direction
 */
void lv_example_bar_7(void)
{
    lv_obj_t * bar_tob = lv_bar_create(lv_screen_active());
    lv_obj_set_size(bar_tob, 20, 180);
    lv_bar_set_range(bar_tob, 100, 0);
    lv_bar_set_value(bar_tob, 70, LV_ANIM_OFF);
    lv_obj_align(bar_tob, LV_ALIGN_CENTER, 0, 10);

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "From top to bottom");
    lv_obj_align_to(label, bar_tob, LV_ALIGN_OUT_TOP_MID, 0, -5);
}

#endif
```

6.6.7 API

lv_bar.h

lv_bar_private.h

lv_barcode.h

lv_barcode_private.h

6.7 Button (lv_button)

6.7.1 Panoramica

Buttons have no new features compared to the *Base Widget (lv_obj)*. They are useful for semantic purposes and have slightly different default settings.

Buttons, by default, differ from the Base Widget in the following ways:

- Not scrollable
- Added to the default group
- Default height and width set to `LV_SIZE_CONTENT`

6.7.2 Parts and Styles

- *`LV_PART_MAIN`* The background of the button; uses the *typical background style properties*.

6.7.3 Utilizzo

There are no new features compared to *Base Widget*.

6.7.4 Eventi

- *`LV_EVENT_VALUE_CHANGED`* when the *`LV_OBJ_FLAG_CHECKABLE`* flag is enabled and the Widget is clicked. The event happens on transition to/from the checked state.

Further Reading

Learn more about *Base-Widget Events* emitted by all Widgets.

Learn more about *Eventi*.

6.7.5 Keys

Note that the state of *`LV_KEY_ENTER`* is translated to *`LV_EVENT_PRESSED`*, *`LV_EVENT_PRESSING`* and *`LV_EVENT_RELEASED`* etc.

Further Reading

Learn more about *Keys*.

6.7.6 Esempio

Pulsanti semplici

```
#include "../../lv_examples.h"
#if LV_USE_BUTTON && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);

    if(code == LV_EVENT_CLICKED) {
        LV_LOG_USER("Clicked");
    }
    else if(code == LV_EVENT_VALUE_CHANGED) {
        LV_LOG_USER("Toggled");
    }
}

void lv_example_button_1(void)
{
    lv_obj_t * label;
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_t * btn1 = lv_button_create(lv_screen_active());
    lv_obj_add_event_cb(btn1, event_handler, LV_EVENT_ALL, NULL);
    lv_obj_align(btn1, LV_ALIGN_CENTER, 0, -40);
    lv_obj_remove_flag(btn1, LV_OBJ_FLAG_PRESS_LOCK);

    label = lv_label_create(btn1);
    lv_label_set_text(label, "Button");
    lv_obj_center(label);

    lv_obj_t * btn2 = lv_button_create(lv_screen_active());
    lv_obj_add_event_cb(btn2, event_handler, LV_EVENT_ALL, NULL);
    lv_obj_align(btn2, LV_ALIGN_CENTER, 0, 40);
    lv_obj_add_flag(btn2, LV_OBJ_FLAG_CHECKABLE);
    lv_obj_set_height(btn2, LV_SIZE_CONTENT);

    label = lv_label_create(btn2);
    lv_label_set_text(label, "Toggle");
    lv_obj_center(label);

}
#endif
```

Pulsanti con stili

```
#include "../../lv_examples.h"
#if LV_USE_BUTTON && LV_BUILD_EXAMPLES

/**
 * Style a button from scratch
 */
void lv_example_button_2(void)
{
    /*Init the style for the default state*/
    static lv_style_t style;
    lv_style_init(&style);

    lv_style_set_radius(&style, 3);

    lv_style_set_bg_opa(&style, LV_OPA_100);
    lv_style_set_bg_color(&style, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_bg_grad_color(&style, lv_palette_darken(LV_PALETTE_BLUE,␣
↪2));
    lv_style_set_bg_grad_dir(&style, LV_GRAD_DIR_VER);

    lv_style_set_border_opa(&style, LV_OPA_40);
    lv_style_set_border_width(&style, 2);
    lv_style_set_border_color(&style, lv_palette_main(LV_PALETTE_GREY));

    lv_style_set_shadow_width(&style, 8);
    lv_style_set_shadow_color(&style, lv_palette_main(LV_PALETTE_GREY));
    lv_style_set_shadow_offset_y(&style, 8);

    lv_style_set_outline_opa(&style, LV_OPA_COVER);
    lv_style_set_outline_color(&style, lv_palette_main(LV_PALETTE_BLUE));

    lv_style_set_text_color(&style, lv_color_white());
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_set_pad_all(&style, 10);

    /*Init the pressed style*/
    static lv_style_t style_pr;
    lv_style_init(&style_pr);

    /*Add a large outline when pressed*/
    lv_style_set_outline_width(&style_pr, 30);
    lv_style_set_outline_opa(&style_pr, LV_OPA_TRANSP);

    lv_style_set_translate_y(&style_pr, 5);
    lv_style_set_shadow_offset_y(&style_pr, 3);
    lv_style_set_bg_color(&style_pr, lv_palette_darken(LV_PALETTE_BLUE,␣
↪2));
    lv_style_set_bg_grad_color(&style_pr, lv_palette_darken(LV_PALETTE_
↪BLUE, 4));

    /*Add a transition to the outline*/
    static lv_style_transition_dsc_t trans;
    static lv_style_prop_t props[] = {LV_STYLE_OUTLINE_WIDTH, LV_STYLE_
↪OUTLINE_OPA, 0};
    lv_style_transition_dsc_init(&trans, props, lv_anim_path_linear, 300,␣
↪0, NULL);

    lv_style_set_transition(&style_pr, &trans);

    lv_obj_t * btn1 = lv_button_create(lv_screen_active());
    lv_obj_remove_style_all(btn1);                          /*Remove the␣
↪style coming from the theme*/
    lv_obj_add_style(btn1, &style, 0);
    lv_obj_add_style(btn1, &style_pr, LV_STATE_PRESSED);
    lv_obj_set_size(btn1, LV_SIZE_CONTENT, LV_SIZE_CONTENT);
    lv_obj_center(btn1);

    lv_obj_t * label = lv_label_create(btn1);
    lv_label_set_text(label, "Button");
    lv_obj_center(label);
}
#endif
```

Pulsante gommoso

```
#include "../../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_BUTTON

/**
 * Create a style transition on a button to act like a gum when clicked
 */
void lv_example_button_3(void)
{
    /*Properties to transition*/
    static lv_style_prop_t props[] = {
        LV_STYLE_TRANSFORM_WIDTH, LV_STYLE_TRANSFORM_HEIGHT, LV_STYLE_TEXT_
↪LETTER_SPACE, 0
    };
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Transition descriptor when going back to the default state.
     *Add some delay to be sure the press transition is visible even if␣
↪the press was very short*/
    static lv_style_transition_dsc_t transition_dsc_def;
    lv_style_transition_dsc_init(&transition_dsc_def, props, lv_anim_path_
↪overshoot, 250, 100, NULL);

    /*Transition descriptor when going to pressed state.
     *No delay, go to presses state immediately*/
    static lv_style_transition_dsc_t transition_dsc_pr;
    lv_style_transition_dsc_init(&transition_dsc_pr, props, lv_anim_path_
↪ease_in_out, 250, 0, NULL);

    /*Add only the new transition to he default state*/
    static lv_style_t style_def;
    lv_style_init(&style_def);
    lv_style_set_transition(&style_def, &transition_dsc_def);

    /*Add the transition and some transformation to the presses state.*/
    static lv_style_t style_pr;
    lv_style_init(&style_pr);
    lv_style_set_transform_width(&style_pr, 10);
    lv_style_set_transform_height(&style_pr, -10);
    lv_style_set_text_letter_space(&style_pr, 10);
    lv_style_set_transition(&style_pr, &transition_dsc_pr);

    lv_obj_t * btn1 = lv_button_create(lv_screen_active());
    lv_obj_align(btn1, LV_ALIGN_CENTER, 0, -80);
    lv_obj_add_style(btn1, &style_pr, LV_STATE_PRESSED);
    lv_obj_add_style(btn1, &style_def, 0);

    lv_obj_t * label = lv_label_create(btn1);
    lv_label_set_text(label, "Gum");
}
#endif
```

6.7.7 API

lv_api_map_v9_0.h

lv_button.h

lv_button_private.h

lv_buttonmatrix.h

lv_buttonmatrix_private.h

6.8 Button Matrix (lv_buttonmatrix)

6.8.1 Panoramica

The Button Matrix Widget is a lightweight way to display multiple Buttons in rows and columns --- lightweight because the buttons are not actually created but just virtually drawn on the fly. With Button Matrix, each button uses only eight extra bytes of memory instead of the ~100-150 bytes a normal *Button* Widget plus the 100 or so bytes for the *Label* Widget.

New Button Matrix Widgets are added to the default group (if one is set). Additionally, Button Matrix is an editable Widget: it allows selecting and clicking the buttons with encoder and keyboard navigation as well.

6.8.2 Parts and Styles

- *LV_PART_MAIN* The background of the Button Matrix, uses the typical background style properties. `pad_row` and `pad_column` sets the space between the buttons.
- *LV_PART_ITEMS* The buttons all use the text and typical background style properties except translations and transformations.

6.8.3 Utilizzo

Button map

The number of buttons, their positions and text are controlled by a descriptor string array, called a *map*, passed to *lv_buttonmatrix_set_map*`(btn_matrix, my_map)`. The declaration of a map should look like `const char * map[] = {"button1", "button2", "button3", NULL}`. Note that the last element must be either `NULL` or an empty string (`""`)!

Use `"\n"` in the map to insert a **line break**. E.g. `{"button1", "button2", "\n", "button3", ""}`. Each line's buttons have their width calculated automatically. So in the example the first row will have 2 buttons each with 50% width and a second row with 1 button having 100% width.

> **Nota**
>
> The number of buttons neither includes the newline elements nor the terminating element of the array.

Button widths

The buttons' width can be set in proportion to the width of other buttons in the same row with *lv_buttonmatrix_set_button_width*`(btn_matrix, button_id, width)` E.g. in a line with two buttons: *buttonA, width = 1* and *buttonB, width = 2*, *buttonA* will have 33 % width and *buttonB* will have 66 % width. This is similar to how the "flex-grow" property works in CSS. The width must be in the range [1..15] with the default being 1.

Button behavior

Each button's behavior can be customized with the following control flags:

- *LV_BUTTONMATRIX_CTRL_HIDDEN*: Hides button; it continues to hold its space in layout.
- *LV_BUTTONMATRIX_CTRL_NO_REPEAT*: Do not emit *LV_EVENT_LONG_PRESSED_REPEAT* events while button is long-pressed.
- *LV_BUTTONMATRIX_CTRL_DISABLED*: Disables button like *LV_STATE_DISABLED* on normal Widgets.
- *LV_BUTTONMATRIX_CTRL_CHECKABLE*: Enable toggling of *LV_STATE_CHECKED* when clicked.
- *LV_BUTTONMATRIX_CTRL_CHECKED*: Make the button checked. It will use the *LV_STATE_CHECKED* styles.
- *LV_BUTTONMATRIX_CTRL_CLICK_TRIG*: 1: Enables sending `LV_EVENT_VALUE_CHANGE` on `CLICK`, 0: sends `LV_EVENT_VALUE_CHANGE` on `PRESS`.
- *LV_BUTTONMATRIX_CTRL_POPOVER*: Show button text in a pop-over while being pressed.
- *LV_BUTTONMATRIX_CTRL_RECOLOR*: Enable text recoloring with `#color`
- *LV_BUTTONMATRIX_CTRL_CUSTOM_1*: Custom free-to-use flag
- *LV_BUTTONMATRIX_CTRL_CUSTOM_2*: Custom free-to-use flag

By default, these flags are disabled.

To set and clear a button's control flags, use

- *lv_buttonmatrix_set_button_ctrl*(btn_matrix, button_id, LV_BUTTONMATRIX_CTRL_...) and
- *lv_buttonmatrix_clear_button_ctrl*(btn_matrix, button_id, LV_BUTTONMATRIX_CTRL_...)

respectively. button_id is a zero-based button index (0 = first button). LV_BUTTONMATRIX_CTRL_... values can be bit-wise OR-ed together when passed to these functions.

To set and clear the same control attribute for all buttons in a Button Matrix, use

- *lv_buttonmatrix_set_button_ctrl_all*(btn_matrix, LV_BUTTONMATRIX_CTRL_...) and
- *lv_buttonmatrix_clear_button_ctrl_all*(btn_matrix, LV_BUTTONMATRIX_CTRL_...)

respectively.

To set a control map for a Button Matrix (similar to *Button map*), use

- *lv_buttonmatrix_set_ctrl_map*(btn_matrix, ctrl_map).

An element of ctrl_map should look like ctrl_map[0] = width | *LV_BUTTONMATRIX_CTRL_NO_REPEAT* | *LV_BUTTONMATRIX_CTRL_CHECKABLE*. The number of elements must be equal to the number of buttons.

One checked

The "One-checked" feature can be enabled with *lv_buttonmatrix_set_one_checked*(btn_matrix, true) to allow only one button to be checked at a time.

6.8.4 Eventi

- *LV_EVENT_VALUE_CHANGED*: Sent when a button is pressed/released or repeated after long press. The event parameter is set to the ID of the pressed/released button.

lv_buttonmatrix_get_selected_button(btn_matrix) returns the index of the button most recently released (the button with focus) or LV_BUTTONMATRIX_BUTTON_NONE if no such button was found.

lv_buttonmatrix_get_button_text(btn_matrix, button_id) returns a pointer to the text of the button specified by zero-based index button_id.

Further Reading

Learn more about *Base-Widget Events* emitted by all Widgets.

Learn more about *Eventi*.

6.8.5 Keys

- LV_KEY_RIGHT/UP/LEFT/RIGHT To navigate among the buttons to select one
- *LV_KEY_ENTER* To press/release the selected button

Note that long pressing the Button Matrix with an encoder can mean to enter/leave edit mode and simply long pressing a button to make it repeat as well. To avoid this contradiction, add *lv_buttonmatrix_set_button_ctrl_all*(btn_matrix, *LV_BUTTONMATRIX_CTRL_CLICK_TRIG* | *LV_BUTTONMATRIX_CTRL_NO_REPEAT*) to the Button Matrix if used with an encoder. This disables the repeat feature so the button will not be activated upon leaving edit mode.

Further Reading

Learn more about *Keys*.

6.8.6 Esempio

Matrice di pulsanti semplice

```
#include "../../lv_examples.h"
#if LV_USE_BUTTONMATRIX && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        uint32_t id = lv_buttonmatrix_get_selected_button(obj);
        const char * txt = lv_buttonmatrix_get_button_text(obj, id);
        LV_UNUSED(txt);
        LV_LOG_USER("%s was pressed\n", txt);
    }
}

static const char * btnm_map[] = {"1", "2", "3", "4", "5", "\n",
                                  "6", "7", "8", "9", "0", "\n",
                                  "Action1", "Action2", ""
                                 };

void lv_example_buttonmatrix_1(void)
{
    lv_obj_t * btnm1 = lv_buttonmatrix_create(lv_screen_active());
    lv_buttonmatrix_set_map(btnm1, btnm_map);
    lv_buttonmatrix_set_button_width(btnm1, 10, 2);        /*Make "Action1
↪" twice as wide as "Action2"*/
    lv_buttonmatrix_set_button_ctrl(btnm1, 10, LV_BUTTONMATRIX_CTRL_
↪CHECKABLE);
    lv_buttonmatrix_set_button_ctrl(btnm1, 11, LV_BUTTONMATRIX_CTRL_
↪CHECKED);
    lv_obj_align(btnm1, LV_ALIGN_CENTER, 0, 0);
    lv_obj_add_event_cb(btnm1, event_handler, LV_EVENT_ALL, NULL);
}

#endif
```

Pulsanti personalizzati

```
#include "../../lv_examples.h"
#if LV_USE_BUTTONMATRIX && LV_BUILD_EXAMPLES

static void event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);
    /*When the button matrix draws the buttons...*/
```

(continues on next page)

(continua dalla pagina precedente)

```
    if(base_dsc->part == LV_PART_ITEMS) {
        bool pressed = false;
        if(lv_buttonmatrix_get_selected_button(obj) == base_dsc->id1 && lv_
↪obj_has_state(obj, LV_STATE_PRESSED)) {
            pressed = true;
        }

        /*Change the draw descriptor of the 2nd button*/
        if(base_dsc->id1 == 1) {
            lv_draw_fill_dsc_t * fill_draw_dsc = lv_draw_task_get_fill_
↪dsc(draw_task);
            if(fill_draw_dsc) {
                fill_draw_dsc->radius = 0;
                if(pressed) fill_draw_dsc->color = lv_palette_darken(LV_
↪PALETTE_BLUE, 3);
                else fill_draw_dsc->color = lv_palette_main(LV_PALETTE_
↪BLUE);
            }
            lv_draw_box_shadow_dsc_t * box_shadow_draw_dsc = lv_draw_task_
↪get_box_shadow_dsc(draw_task);
            if(box_shadow_draw_dsc) {
                box_shadow_draw_dsc->width = 6;
                box_shadow_draw_dsc->ofs_x = 3;
                box_shadow_draw_dsc->ofs_y = 3;
            }
            lv_draw_label_dsc_t * label_draw_dsc = lv_draw_task_get_label_
↪dsc(draw_task);
            if(label_draw_dsc) {
                label_draw_dsc->color = lv_color_white();
            }

        }
        /*Change the draw descriptor of the 3rd button*/
        else if(base_dsc->id1 == 2) {
            lv_draw_fill_dsc_t * fill_draw_dsc = lv_draw_task_get_fill_
↪dsc(draw_task);
            if(fill_draw_dsc) {
                fill_draw_dsc->radius = LV_RADIUS_CIRCLE;
                if(pressed) fill_draw_dsc->color = lv_palette_darken(LV_
↪PALETTE_RED, 3);
                else fill_draw_dsc->color = lv_palette_main(LV_PALETTE_
↪RED);
            }

            lv_draw_box_shadow_dsc_t * box_shadow_draw_dsc = lv_draw_task_
↪get_box_shadow_dsc(draw_task);
            if(box_shadow_draw_dsc) {
                box_shadow_draw_dsc->radius = LV_RADIUS_CIRCLE;
            }
        }
        else if(base_dsc->id1 == 3) {
            lv_draw_label_dsc_t * label_draw_dsc = lv_draw_task_get_label_
↪dsc(draw_task);
            if(label_draw_dsc) {
                label_draw_dsc->opa = 0;
            }
```

(continues on next page)

(continua dalla pagina precedente)

```
            if(lv_draw_task_get_type(draw_task) == LV_DRAW_TASK_TYPE_FILL)
↪{
                LV_IMAGE_DECLARE(img_star);
                lv_image_header_t header;
                lv_result_t res = lv_image_decoder_get_info(&img_star, &
↪header);
                if(res != LV_RESULT_OK) return;

                lv_area_t a;
                a.x1 = 0;
                a.x2 = header.w - 1;
                a.y1 = 0;
                a.y2 = header.h - 1;
                lv_area_t draw_task_area;
                lv_draw_task_get_area(draw_task, &draw_task_area);
                lv_area_align(&draw_task_area, &a, LV_ALIGN_CENTER, 0, 0);

                lv_draw_image_dsc_t img_draw_dsc;
                lv_draw_image_dsc_init(&img_draw_dsc);
                img_draw_dsc.src = &img_star;
                img_draw_dsc.recolor = lv_color_black();
                if(pressed) img_draw_dsc.recolor_opa = LV_OPA_30;

                lv_draw_image(base_dsc->layer, &img_draw_dsc, &a);

            }
        }
    }
}

/**
 * Add custom drawer to the button matrix to customize buttons one by one
 */
void lv_example_buttonmatrix_2(void)
{
    lv_obj_t * btnm = lv_buttonmatrix_create(lv_screen_active());
    lv_obj_add_event_cb(btnm, event_cb, LV_EVENT_DRAW_TASK_ADDED, NULL);
    lv_obj_add_flag(btnm, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);
    lv_obj_center(btnm);
}

#endif
```

Paginazione

```
#include "../../lv_examples.h"
#if LV_USE_BUTTONMATRIX  && LV_BUILD_EXAMPLES

static void event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    uint32_t id = lv_buttonmatrix_get_selected_button(obj);
    bool prev = id == 0;
    bool next = id == 6;
    if(prev || next) {
        /*Find the checked button*/
```

(continues on next page)

(continua dalla pagina precedente)

```
        uint32_t i;
        for(i = 1; i < 7; i++) {
            if(lv_buttonmatrix_has_button_ctrl(obj, i, LV_BUTTONMATRIX_
↪CTRL_CHECKED)) break;
        }

        if(prev && i > 1) i--;
        else if(next && i < 5) i++;

        lv_buttonmatrix_set_button_ctrl(obj, i, LV_BUTTONMATRIX_CTRL_
↪CHECKED);
    }
}

/**
 * Make a button group (pagination)
 */
void lv_example_buttonmatrix_3(void)
{
    static lv_style_t style_bg;
    lv_style_init(&style_bg);
    lv_style_set_pad_all(&style_bg, 0);
    lv_style_set_pad_gap(&style_bg, 0);
    lv_style_set_clip_corner(&style_bg, true);
    lv_style_set_radius(&style_bg, LV_RADIUS_CIRCLE);
    lv_style_set_border_width(&style_bg, 0);

    static lv_style_t style_btn;
    lv_style_init(&style_btn);
    lv_style_set_radius(&style_btn, 0);
    lv_style_set_border_width(&style_btn, 1);
    lv_style_set_border_opa(&style_btn, LV_OPA_50);
    lv_style_set_border_color(&style_btn, lv_palette_main(LV_PALETTE_
↪GREY));
    lv_style_set_border_side(&style_btn, LV_BORDER_SIDE_INTERNAL);
    lv_style_set_radius(&style_btn, 0);

    static const char * map[] = {LV_SYMBOL_LEFT, "1", "2", "3", "4", "5",␣
↪LV_SYMBOL_RIGHT, ""};

    lv_obj_t * btnm = lv_buttonmatrix_create(lv_screen_active());
    lv_buttonmatrix_set_map(btnm, map);
    lv_obj_add_style(btnm, &style_bg, 0);
    lv_obj_add_style(btnm, &style_btn, LV_PART_ITEMS);
    lv_obj_add_event_cb(btnm, event_cb, LV_EVENT_VALUE_CHANGED, NULL);
    lv_obj_set_size(btnm, 225, 35);

    /*Allow selecting on one number at time*/
    lv_buttonmatrix_set_button_ctrl_all(btnm, LV_BUTTONMATRIX_CTRL_
↪CHECKABLE);
    lv_buttonmatrix_clear_button_ctrl(btnm, 0, LV_BUTTONMATRIX_CTRL_
↪CHECKABLE);
    lv_buttonmatrix_clear_button_ctrl(btnm, 6, LV_BUTTONMATRIX_CTRL_
↪CHECKABLE);

    lv_buttonmatrix_set_one_checked(btnm, true);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_buttonmatrix_set_button_ctrl(btnm, 1, LV_BUTTONMATRIX_CTRL_CHECKED);

    lv_obj_center(btnm);

}

#endif
```

6.8.7 API

lv_buttonmatrix.h

lv_buttonmatrix_private.h

6.9 Calendar (lv_calendar)

6.9.1 Panoramica

The Calendar Widget is a classic calendar which can:

- show the days of any month in a 7x7 matrix;
- show the name of the days;
- highlight the current day (today);
- highlight any user-defined dates.

The Calendar is added to the default group (if one is set). Calendar is an editable Widget which allows selecting and clicking the dates with encoder or keyboard navigation as well as pointer-type input devices.

To make the Calendar flexible, by default it does not show the current year or month. Instead, there are optional "headers" that can be attached to the calendar.

6.9.2 Parts and Styles

The calendar Widget uses the *Button Matrix* Widget under the hood to arrange the days into a matrix.

- `LV_PART_MAIN` Calendar background. Uses the *typical background style properties*.
- `LV_PART_ITEMS` Refers to dates and day names. Button matrix control flags are set to differentiate the buttons and a custom drawer event is added to modify the properties of the buttons as follows:
  - day names have no border, no background and are drawn with a gray color
  - days of the previous and next month have the `LV_BUTTONMATRIX_CTRL_DISABLED` flag
  - today has a thicker border with the theme's primary color - highlighted days have some opacity with the theme's primary color.

6.9.3 Utilizzo

Some functions use the `lv_calendar_date_t` type which is a structure with `year`, `month` and `day` fields.

Current date

To set the current date (today), use the `lv_calendar_set_today_date(calendar, year, month, day)` function. `month` needs to be in 1..12 range and `day` in 1..31 range.

Month shown

To set the shown date, use `lv_calendar_set_month_shown(calendar, year, month)`

Highlighted days

The list of highlighted dates should be stored in a `lv_calendar_date_t` array and applied to the Calendar by calling `lv_calendar_set_highlighted_dates(calendar, highlighted_dates, date_num)`. Only the array's pointer will be saved so the array should be have static or global scope.

Names of days

The names of the days can be adjusted with `lv_calendar_set_day_names(calendar, day_names)` where `day_names` looks like `const char * day_names[7] = {"Su", "Mo", ...};` Only the pointer of the day names is saved so the array should have static or global scope.

Custom year list

Set a custom year list with `lv_calendar_header_dropdown_set_year_list(calendar, years_list)` where `years_list` is a pointer to the custom years list. It can be a constant string like `static const char * years = "2023\n2022\n2021\n2020\n2019";`, or can be generated dynamically into a buffer as well. Calendar stores these in a Drop-Down List Widget via `lv_dropdown_set_options()` so the passed string pointer can be supplied by a local variable or buffer and does not need to persist beyond the call.

Calendario cinese

The Chinese calendar is a traditional cultural tool that integrates elements such as the lunar calendar, solar terms, and traditional festivals. It is widely used in Chinese social life, helping people understand the dates of the lunar calendar, arrange festival activities, and inherit the excellent traditional culture of the Chinese nation. Whether in families, businesses, or education, the Chinese calendar plays an irreplaceable role, enabling people to better understand and appreciate the charm of Chinese traditional culture.

If you want to use the Chinese calendar, please use `lv_calendar_set_chinese_mode(calendar, true)` to enable it.

6.9.4 Eventi

- `LV_EVENT_VALUE_CHANGED` Sent if a date is clicked. `lv_calendar_get_pressed_date(calendar, &date)` to set `date` to the date currently being pressed. Returns `LV_RESULT_OK` if there is a valid pressed date; otherwise it returns `LV_RESULT_INVALID`.

> **Further Reading**
>
> Learn more about *Base-Widget Events* emitted by all Widgets.
>
> Learn more about *Eventi*.

6.9.5 Keys

- `LV_KEY_DOWN/UP/LEFT/RIGHT` To navigate among the buttons to dates
- `LV_KEY_ENTER` To press/release the selected date

> **Further Reading**
>
> Learn more about *Keys*.

6.9.6 Headers

From LVGL v8.1 onward, the header is added directly into the Calendar Widget and the API of the headers has been changed.

Arrow buttons

`lv_calendar_add_header_arrow(calendar)` creates a header that contains a left and right arrow on the sides and text between the arrows showing the current year and month.

Drop-down

`lv_calendar_add_header_dropdown(calendar)` creates a header that contains 2 Drop-Drown List Widgets for the year and month.

6.9.7 Esempio

Calendario con intestazione

```
#include "../../lv_examples.h"
#if LV_USE_CALENDAR && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = (lv_obj_t *)lv_event_get_current_target(e);

    if(code == LV_EVENT_VALUE_CHANGED) {
        lv_calendar_date_t date;
        if(lv_calendar_get_pressed_date(obj, &date)) {
            LV_LOG_USER("Clicked date: %02d.%02d.%d", date.day, date.month,
↪ date.year);
        }
    }
}

void lv_example_calendar_1(void)
{
    lv_obj_t  * calendar = lv_calendar_create(lv_screen_active());
    lv_obj_set_size(calendar, 185, 230);
    lv_obj_align(calendar, LV_ALIGN_CENTER, 0, 27);
    lv_obj_add_event_cb(calendar, event_handler, LV_EVENT_ALL, NULL);

    lv_calendar_set_today_date(calendar, 2021, 02, 23);
    lv_calendar_set_month_shown(calendar, 2021, 02);

    /*Highlight a few days*/
    static lv_calendar_date_t highlighted_days[3];       /*Only its␣
↪pointer will be saved so should be static*/
    highlighted_days[0].year = 2021;
    highlighted_days[0].month = 02;
    highlighted_days[0].day = 6;

    highlighted_days[1].year = 2021;
    highlighted_days[1].month = 02;
    highlighted_days[1].day = 11;

    highlighted_days[2].year = 2022;
```

(continues on next page)

(continua dalla pagina precedente)

```
    highlighted_days[2].month = 02;
    highlighted_days[2].day = 22;

    lv_calendar_set_highlighted_dates(calendar, highlighted_days, 3);

#if LV_USE_CALENDAR_HEADER_DROPDOWN
    lv_calendar_add_header_dropdown(calendar);
#elif LV_USE_CALENDAR_HEADER_ARROW
    lv_calendar_add_header_arrow(calendar);
#endif
}

#endif
```

Calendario cinese

```
#include "../../lv_examples.h"
#if LV_USE_CALENDAR && LV_USE_CALENDAR_CHINESE && LV_BUILD_EXAMPLES

void lv_example_calendar_2(void)
{
    lv_obj_t  * calendar = lv_calendar_create(lv_screen_active());
    lv_obj_set_size(calendar, 300, 300);
    lv_obj_align(calendar, LV_ALIGN_TOP_MID, 0, 0);

    lv_calendar_set_today_date(calendar, 2024, 03, 22);
    lv_calendar_set_month_shown(calendar, 2024, 03);

#if LV_USE_CALENDAR_HEADER_DROPDOWN
    lv_calendar_add_header_dropdown(calendar);
#elif LV_USE_CALENDAR_HEADER_ARROW
    lv_calendar_add_header_arrow(calendar);
#endif

    lv_calendar_set_chinese_mode(calendar, true);
    lv_obj_set_style_text_font(calendar, &lv_font_source_han_sans_sc_14_
↪cjk, LV_PART_MAIN);
}

#else

void lv_example_calendar_2(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "chinese calendar is not enabled");
    lv_obj_center(label);
}

#endif
```

6.9.8 API

lv_api_map_v9_1.h

lv_calendar.h

lv_calendar_chinese.h

lv_calendar_header_arrow.h

lv_calendar_header_dropdown.h

lv_calendar_private.h

6.10 Canvas (lv_canvas)

6.10.1 Panoramica

A Canvas inherits from *Image* and extends it, enabling the user to draw anything. Rectangles, text, images, lines, arcs, etc. can be drawn here using LVGL's extensive drawing engine.

6.10.2 Parts and Styles

- `LV_PART_MAIN` Uses the *typical background* and image style properties.

6.10.3 Utilizzo

Buffer

The Canvas needs a buffer in which to store the drawn image. To assign a buffer to a Canvas, use `lv_canvas_set_buffer(canvas, buffer, width, height, LV_COLOR_FORMAT_...)`. Where `buffer` is a static buffer (not just a local variable) to hold the image of the Canvas. For example, for a 100x50 ARGB8888 buffer: `static uint8_t buffer[100 * 50 * 4]`.

Or you can use `static uint8_t buffer[LV_CANVAS_BUF_SIZE(width, height, bits_per_pixel, stride_in_bytes)]`.

Canvas supports all the color formats like `LV_COLOR_FORMAT_ARGB8888` or `LV_COLOR_FORMAT_I2`. See the full list in the *Color formats* section.

Indexed colors

For indexed color formats (`LV_COLOR_FORMAT_I1/2/4/8`), the palette needs to be populated for all palette indices that will be used using `lv_canvas_set_palette(canvas, index, color)`. For example, the following sets pixels with *index==3* to red.

```
lv_canvas_set_palette(canvas, 3, lv_color_hex(0xff0000))
```

Drawing

To set an individual pixel's color on the Canvas, use `lv_canvas_set_px(canvas, x, y, color, opa)`. With indexed color formats (`LV_COLOR_FORMAT_I1/2/4/8`) pass the color index as the `color` argument by using the *blue* channel in the `color` value, e.g. `lv_color_make(0, 0, index)`.

`lv_canvas_fill_bg(canvas, lv_color_hex(0x00ff00), LV_OPA_50)` fills the whole Canvas to blue with 50% opacity. Note that if the current color format doesn't support colors (e.g. `LV_COLOR_FORMAT_A8`) the color will be ignored. Similarly, if opacity is not supported (e.g. `LV_COLOR_FORMAT_RGB565`), it will be ignored.

An array of pixels can be copied to the Canvas with `lv_canvas_copy_buf(canvas, buffer_to_copy, x, y, width, height)`. The color format of the buffer and Canvas need to match.

To draw something to the Canvas use LVGL's draw functions directly. See the examples for more details.

The draw functions can draw to any color format to which LVGL can render. Typically this means `LV_COLOR_FORMAT_RGB565`, `LV_COLOR_FORMAT_RGB888`, `LV_COLOR_FORMAT_XRGB888`, and `LV_COLOR_FORMAT_ARGB8888`.

6.10.4 Eventi

No special events are sent by Canvas Widgets.

Further Reading

Learn more about *Base-Widget Events* emitted by all Widgets.

Learn more about *Eventi*.

6.10.5 Keys

No *Keys* are processed by Canvas Widgets.

Further Reading

Learn more about *Keys*.

6.10.6 Esempio

Disegno sulla Canvas e rotazione

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  200
#define CANVAS_HEIGHT  150

void lv_example_canvas_1(void)
{
    lv_draw_rect_dsc_t rect_dsc;
    lv_draw_rect_dsc_init(&rect_dsc);
    rect_dsc.radius = 10;
    rect_dsc.bg_opa = LV_OPA_COVER;
    rect_dsc.bg_grad.dir = LV_GRAD_DIR_VER;
    rect_dsc.bg_grad.stops[0].color = lv_palette_main(LV_PALETTE_RED);
    rect_dsc.bg_grad.stops[0].opa = LV_OPA_100;
    rect_dsc.bg_grad.stops[1].color = lv_palette_main(LV_PALETTE_BLUE);
    rect_dsc.bg_grad.stops[1].opa = LV_OPA_50;
    rect_dsc.border_width = 2;
    rect_dsc.border_opa = LV_OPA_90;
    rect_dsc.border_color = lv_color_white();
    rect_dsc.shadow_width = 5;
    rect_dsc.shadow_offset_x = 5;
    rect_dsc.shadow_offset_y = 5;

    lv_draw_label_dsc_t label_dsc;
    lv_draw_label_dsc_init(&label_dsc);
    label_dsc.color = lv_palette_main(LV_PALETTE_ORANGE);
    label_dsc.text = "Some text on text canvas";
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf_16bpp, CANVAS_WIDTH, CANVAS_HEIGHT,␣
↪LV_COLOR_FORMAT_RGB565);
    LV_DRAW_BUF_INIT_STATIC(draw_buf_16bpp);

    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_canvas_set_draw_buf(canvas, &draw_buf_16bpp);
    lv_obj_center(canvas);
    lv_canvas_fill_bg(canvas, lv_palette_lighten(LV_PALETTE_GREY, 3), LV_
↪OPA_COVER);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_area_t coords_rect = {30, 20, 100, 70};
    lv_draw_rect(&layer, &rect_dsc, &coords_rect);

    lv_area_t coords_text = {40, 80, 100, 120};
    lv_draw_label(&layer, &label_dsc, &coords_text);

    lv_canvas_finish_layer(canvas, &layer);

    /*Test the rotation. It requires another buffer where the original␣
↪image is stored.
     *So use previous canvas as image and rotate it to the new canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf_32bpp, CANVAS_WIDTH, CANVAS_HEIGHT,␣
↪LV_COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf_32bpp);

    /*Create a canvas and initialize its palette*/
    canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf_32bpp);
    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_canvas_fill_bg(canvas, lv_palette_lighten(LV_PALETTE_GREY, 1), LV_
↪OPA_COVER);

    lv_canvas_init_layer(canvas, &layer);
    lv_image_dsc_t img;
    lv_draw_buf_to_image(&draw_buf_16bpp, &img);
    lv_draw_image_dsc_t img_dsc;
    lv_draw_image_dsc_init(&img_dsc);
    img_dsc.rotation = 120;
    img_dsc.src = &img;
    img_dsc.pivot.x = CANVAS_WIDTH / 2;
    img_dsc.pivot.y = CANVAS_HEIGHT / 2;

    lv_area_t coords_img = {0, 0, CANVAS_WIDTH - 1, CANVAS_HEIGHT - 1};
    lv_draw_image(&layer, &img_dsc, &coords_img);

    lv_canvas_finish_layer(canvas, &layer);
}

#endif
```

Canvas trasparente con chroma key

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  80
```

(continues on next page)

(continua dalla pagina precedente)

```
#define CANVAS_HEIGHT  40

/**
 * Create a transparent canvas with transparency
 */
void lv_example_canvas_2(void)
{
    lv_obj_set_style_bg_color(lv_screen_active(), lv_palette_lighten(LV_
↪PALETTE_RED, 5), 0);

    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf);
    lv_obj_center(canvas);

    /*Red background (There is no dedicated alpha channel in indexed␣
↪images so LV_OPA_COVER is ignored)*/
    lv_canvas_fill_bg(canvas, lv_palette_main(LV_PALETTE_BLUE), LV_OPA_
↪COVER);

    /*Create hole on the canvas*/
    int32_t x;
    int32_t y;
    for(y = 10; y < 20; y++) {
        for(x = 5; x < 75; x++) {
            lv_canvas_set_px(canvas, x, y, lv_palette_main(LV_PALETTE_
↪BLUE), LV_OPA_50);
        }
    }

    for(y = 20; y < 30; y++) {
        for(x = 5; x < 75; x++) {
            lv_canvas_set_px(canvas, x, y, lv_palette_main(LV_PALETTE_
↪BLUE), LV_OPA_20);
        }
    }

    for(y = 30; y < 40; y++) {
        for(x = 5; x < 75; x++) {
            lv_canvas_set_px(canvas, x, y, lv_palette_main(LV_PALETTE_
↪BLUE), LV_OPA_0);
        }
    }
}
#endif
```

Disegnare un rettangolo sul canvas

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES
```

(continues on next page)

(continua dalla pagina precedente)

```
#define CANVAS_WIDTH  50
#define CANVAS_HEIGHT  50

/**
 * Draw a rectangle to the canvas
 */
void lv_example_canvas_3(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf);

    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_draw_rect_dsc_t dsc;
    lv_draw_rect_dsc_init(&dsc);
    dsc.bg_color = lv_palette_main(LV_PALETTE_RED);
    dsc.border_color = lv_palette_main(LV_PALETTE_BLUE);
    dsc.border_width = 3;
    dsc.outline_color = lv_palette_main(LV_PALETTE_GREEN);
    dsc.outline_width = 2;
    dsc.outline_pad = 2;
    dsc.outline_opa = LV_OPA_50;
    dsc.radius = 5;
    dsc.border_width = 3;

    lv_area_t coords = {10, 10, 40, 30};

    lv_draw_rect(&layer, &dsc, &coords);

    lv_canvas_finish_layer(canvas, &layer);
}
#endif
```

Disegnare un'etichetta sul canvas

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_FONT_MONTSERRAT_18 && LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  50
#define CANVAS_HEIGHT  50

/**
 * Draw a text to the canvas
 */
void lv_example_canvas_4(void)
{
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf);
    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_draw_label_dsc_t dsc;
    lv_draw_label_dsc_init(&dsc);
    dsc.color = lv_palette_main(LV_PALETTE_RED);
    dsc.font = &lv_font_montserrat_18;
    dsc.decor = LV_TEXT_DECOR_UNDERLINE;
    dsc.text = "Hello";

    lv_area_t coords = {10, 10, 30, 60};

    lv_draw_label(&layer, &dsc, &coords);

    lv_canvas_finish_layer(canvas, &layer);

}
#endif
```

Disegnare un arco sul canvas

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  50
#define CANVAS_HEIGHT  50

/**
 * Draw an arc to the canvas
 */
void lv_example_canvas_5(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf);
    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_draw_arc_dsc_t dsc;
    lv_draw_arc_dsc_init(&dsc);
    dsc.color = lv_palette_main(LV_PALETTE_RED);
    dsc.width = 5;
    dsc.center.x = 25;
    dsc.center.y = 25;
    dsc.width = 10;
    dsc.radius = 15;
    dsc.start_angle = 0;
    dsc.end_angle = 220;

    lv_draw_arc(&layer, &dsc);

    lv_canvas_finish_layer(canvas, &layer);

}
#endif
```

Disegnare un'immagine sul canvas

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  50
#define CANVAS_HEIGHT  50

/**
 * Draw an image to the canvas
 */
void lv_example_canvas_6(void)
{
    /*Create a buffer for the canvas*/
    static uint8_t cbuf[LV_CANVAS_BUF_SIZE(CANVAS_WIDTH, CANVAS_HEIGHT, 32,
↪ LV_DRAW_BUF_STRIDE_ALIGN)];

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_buffer(canvas, cbuf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    LV_IMAGE_DECLARE(img_star);
    lv_draw_image_dsc_t dsc;
    lv_draw_image_dsc_init(&dsc);
    dsc.src = &img_star;

    lv_area_t coords = {10, 10, 10 + img_star.header.w - 1, 10 + img_star.
↪header.h - 1};

    lv_draw_image(&layer, &dsc, &coords);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_canvas_finish_layer(canvas, &layer);
}
#endif
```

Disegnare una linea sul canvas

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS&& LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  50
#define CANVAS_HEIGHT  50

/**
 * Draw a line to the canvas
 */
void lv_example_canvas_7(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf);
    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_draw_line_dsc_t dsc;
    lv_draw_line_dsc_init(&dsc);
    dsc.color = lv_palette_main(LV_PALETTE_RED);
    dsc.width = 4;
    dsc.round_end = 1;
    dsc.round_start = 1;
    dsc.p1.x = 15;
    dsc.p1.y = 15;
    dsc.p2.x = 35;
    dsc.p2.y = 10;
    lv_draw_line(&layer, &dsc);

    lv_canvas_finish_layer(canvas, &layer);

}
#endif
```

Disegnare un'immagine vettoriale sul canvas

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES

#if LV_USE_VECTOR_GRAPHIC
```

(continues on next page)

(continua dalla pagina precedente)

```
#define CANVAS_WIDTH  150
#define CANVAS_HEIGHT 150

/**
 * Draw a path to the canvas
 */
void lv_example_canvas_8(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf);
    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_vector_dsc_t * dsc = lv_vector_dsc_create(&layer);
    lv_vector_path_t * path = lv_vector_path_create(LV_VECTOR_PATH_QUALITY_
↪MEDIUM);

    lv_fpoint_t pts[] = {{10, 10}, {130, 130}, {10, 130}};
    lv_vector_path_move_to(path, &pts[0]);
    lv_vector_path_line_to(path, &pts[1]);
    lv_vector_path_line_to(path, &pts[2]);
    lv_vector_path_close(path);

    lv_vector_dsc_set_fill_color(dsc, lv_color_make(0x00, 0x80, 0xff));
    lv_vector_dsc_add_path(dsc, path);

    lv_draw_vector(dsc);
    lv_vector_path_delete(path);
    lv_vector_dsc_delete(dsc);

    lv_canvas_finish_layer(canvas, &layer);
}
#else

void lv_example_canvas_8(void)
{
    /*fallback for online examples*/
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Vector graphics is not enabled");
    lv_obj_center(label);
}

#endif /*LV_USE_VECTOR_GRAPHIC*/

#endif
```

Disegnare un triangolo sul canvas

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES


#define CANVAS_WIDTH  150
#define CANVAS_HEIGHT 150

/**
 * Draw a triangle to the canvas
 */
void lv_example_canvas_9(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf);
    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_draw_triangle_dsc_t tri_dsc;
    lv_draw_triangle_dsc_init(&tri_dsc);
    tri_dsc.p[0].x = 10;
    tri_dsc.p[0].y = 10;
    tri_dsc.p[1].x = 100;
    tri_dsc.p[1].y = 30;
    tri_dsc.p[2].x = 50;
    tri_dsc.p[2].y = 100;

    tri_dsc.grad.stops_count = 2;
    tri_dsc.grad.dir = LV_GRAD_DIR_VER;
    tri_dsc.grad.stops[0].color = lv_color_hex(0xff0000);
    tri_dsc.grad.stops[0].frac = 64;    /*Start at 25%*/
    tri_dsc.grad.stops[0].opa = LV_OPA_COVER;
    tri_dsc.grad.stops[1].color = lv_color_hex(0x0000ff);
    tri_dsc.grad.stops[1].opa = LV_OPA_TRANSP;
    tri_dsc.grad.stops[1].frac = 3 * 64;    /*End at 75%*/

    tri_dsc.opa = 128;  /*Set the overall opacity to 50%*/

    lv_draw_triangle(&layer, &tri_dsc);

    lv_canvas_finish_layer(canvas, &layer);
}

#endif
```

Disegnare Effetti di Lettere Fantasiose

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  300
#define CANVAS_HEIGHT  200

static void timer_cb(lv_timer_t * timer)
{
    static int16_t counter = 0;
    const char * string = "lol~ I'm wavvvvvvving~>>>";
    const int16_t string_len = lv_strlen(string);

    lv_obj_t * canvas = (lv_obj_t *)lv_timer_get_user_data(timer);
    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_canvas_fill_bg(canvas, lv_color_white(), LV_OPA_COVER);

    lv_draw_letter_dsc_t letter_dsc;
    lv_draw_letter_dsc_init(&letter_dsc);
    letter_dsc.color = lv_color_hex(0xff0000);
    letter_dsc.font = lv_font_get_default();

    {
#define CURVE2_X(t) (t * 2 + 10)
#define CURVE2_Y(t) (lv_trigo_sin((t) * 5) * 40 / 32767 + CANVAS_HEIGHT /␣
↪2)

        int32_t pre_x = CURVE2_X(-1);
        int32_t pre_y = CURVE2_Y(-1);
        for(int16_t i = 0; i < string_len; i++) {
            const int16_t angle = (int16_t)(i * 5);
            const int32_t x = CURVE2_X(angle);
            const int32_t y = CURVE2_Y(angle + counter / 2);
            const lv_point_t point = { .x = x, .y = y };

            letter_dsc.unicode = (uint32_t)string[i % string_len];
            letter_dsc.rotation = lv_atan2(y - pre_y, x - pre_x) * 10;
            letter_dsc.color = lv_color_hsv_to_rgb(i * 10, 100, 100);
            lv_draw_letter(&layer, &letter_dsc, &point);

            pre_x = x;
            pre_y = y;
        }
    }

    lv_canvas_finish_layer(canvas, &layer);

    counter++;
}

void lv_example_canvas_10(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
```

(continues on next page)

(continua dalla pagina precedente)

```
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_obj_set_size(canvas, CANVAS_WIDTH, CANVAS_HEIGHT);

    lv_obj_center(canvas);

    lv_canvas_set_draw_buf(canvas, &draw_buf);

    lv_timer_create(timer_cb, 16, canvas);
}

#endif
```

Disegnare Effetti di Lettere Fantasiose 2

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  300
#define CANVAS_HEIGHT  200

static void timer_cb(lv_timer_t * timer)
{
    static int32_t counter = 0;
    const char * string = "windstorrrrrrrrrrrrrrrm~>>>";
    const int16_t string_len = lv_strlen(string);

    lv_obj_t * canvas = (lv_obj_t *) lv_timer_get_user_data(timer);
    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_canvas_fill_bg(canvas, lv_color_white(), LV_OPA_COVER);

    lv_draw_letter_dsc_t letter_dsc;
    lv_draw_letter_dsc_init(&letter_dsc);
    letter_dsc.color = lv_color_hex(0xff0000);
    letter_dsc.font = lv_font_get_default();

    {
#define CURVE2_X(t) ((t) * 2 + lv_trigo_cos((t) * 5) * 40 / 32767 - 10)
#define CURVE2_Y(t, T) ((t) * lv_trigo_sin(((t) + (T)) * 5) * 40 / 32767 /␣
↪80 + CANVAS_HEIGHT / 2)

        int32_t pre_x = CURVE2_X(-1);
        int32_t pre_y = CURVE2_Y(-1, 0);
        for(int16_t i = 0; i < string_len; i++) {
            const int32_t angle = i * 5;
            const int32_t x = CURVE2_X(angle);
            const int32_t y = CURVE2_Y(angle + 30, counter / 2);

            letter_dsc.unicode = (uint32_t)string[i % string_len];
            letter_dsc.rotation = lv_atan2(y - pre_y, x - pre_x) * 10;
            letter_dsc.color = lv_color_hsv_to_rgb(i * 10, 100, 100);
            lv_point_t p = (lv_point_t) {
```

(continues on next page)

(continua dalla pagina precedente)

```
                .x = x, .y = y
            };
            lv_draw_letter(&layer, &letter_dsc, &p);

            pre_x = x;
            pre_y = y;
        }
    }

    lv_canvas_finish_layer(canvas, &layer);

    counter++;
}

void lv_example_canvas_11(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_obj_set_size(canvas, CANVAS_WIDTH, CANVAS_HEIGHT);

    lv_obj_center(canvas);

    lv_canvas_set_draw_buf(canvas, &draw_buf);

    lv_timer_create(timer_cb, 16, canvas);
}

#endif
```

6.10.7 API

lv_canvas.h

lv_canvas_private.h

6.11 Chart (lv_chart)

6.11.1 Panoramica

Chart Widgets are used to visualize data.

Charts have:

- 0 or more background division lines (horizontal and vertical),
- 4 internal axes, any of which can be used to specify scale for a data series,
- a `point_count` (default 10) that applies to all data series added,
- 0 or more data series (you can add or remove them at any time),
- 0 or more cursors (you can add or remove them at any time),
- update mode (modifies behavior of *`lv_chart_set_next_value()`* if you use it).

Charts can:

- display flexible axes with ticks and text
- show precise locations of points or other locations on chart with cursors
- show or hide individual data series
- show or hide individual data points
- show or hide cursors
- scroll and zoom

6.11.2 Chart Basics

Each chart has the following attributes (over and above attributes found in *all Widgets*):

Type (governs how a chart's data series are drawn)

- Can be LINE (default), BAR, SCATTER, or none.
- You can change the chart's type at any point during its life.

Horizontal and Vertical division lines

- default 3 and 5 respectively
- can be any non-negative value including 0

2 Y axes and 2 X axes (the latter are used with SCATTER charts)

- All 4 axes come with each chart automatically (they do not have to be created).
- Their default ranges are [0..100]. If you need a different range, set it before chart is drawn.
- You "use" an axis by associating it with a data series, which happens when the data series is created (more on this below). More than one data series can be associated with each axis.

Point count (number of data points in all data series added to the chart)

- default 10
- If you provide your own data-value arrays, each array so provided must contain at least this number of values.
- For LINE- and BAR-charts, this is the number of points on the X axis.
- LINE- and BAR-charts require only one data-value array to supply Y-values for each data point.
- For SCATTER charts, this is the number of scatter-points in the data series.
- SCATTER charts have separate data-value arrays for both X-values and Y-values.

Any number of data series

- After a chart is created, it initially contains no data series. You have to add them.
- You can add and remove data series at any time during a chart's life.
- When a data series is created, it comes with pre-allocated values array(s) based on its chart type and `point_count`. (All chart types use an array of Y-values. SCATTER-type charts also use an array of X-values.). All Y-values so allocated are set to `LV_CHART_POINT_NONE`, which causes that point to be hidden.
- To get points to be drawn on the chart, you must set their Y-values to something other than `LV_CHART_POINT_NONE`.
- You can hide a point by setting its Y-value to `LV_CHART_POINT_NONE`.
- If desired, you can tell a data series to instead use a value array you provide. If you do:
  - Pre-allocated value arrays are automatically freed.
  - That data series will continue to use *your* array from that time onward.
  - The values in your array must remain available through the life of that data series.

  - You must ensure each array provided contains at least `point_count` `int32_t` elements.
  - Management of the life any value arrays you provide is up to you.

Any number of cursors

- After a chart is created, it initially contains no cursors. You have to add them if you want to use them.
- You can add, show, hide or remove cursors at any time during a chart's life.

Update mode

- *See below*

Chart layers

When a chart is drawn, certain things appear on top of other things, in this order, from back to front:

- The chart's background (with optional division lines)
- Each data series:
  - Earliest data series added appears on top.
  - For a SCATTER chart, within each series, points later in the sequence will appear on top of points earlier in the sequence when there is overlap.
- Each cursor (if there are any):
  - The most recent cursor added appears on top.

6.11.3 Parts and Styles

- `LV_PART_MAIN` The background of the chart. Uses the *typical background* and line style properties (for division lines). *Padding* makes the series area smaller. For BAR charts `pad_column` sets the space between bars in the same data series.
- `LV_PART_SCROLLBAR` A scrollbar used if the chart is zoomed. See *Base Widget (lv_obj)*'s documentation for details.
- `LV_PART_ITEMS` Refers to the LINE or BAR data series.
  - LINE chart: *line* properties are used by lines. `width`, `height`, `bg_color` and `radius` are used to set the appearance of points on the line.
  - Bar chart: The typical background properties are used to style the bars. `pad_column` sets the space between columns in the same data series.
- `LV_PART_INDICATOR` Refers to points on LINE- and SCATTER-charts (small circles or squares [with possibly-rounded corners]).
- `LV_PART_CURSOR` *Line* properties are used to style cursors. `width`, `height`, `bg_color` and `radius` are used to set the appearance of the cursor's "point" showing its location. If either `width` or `height` are set to 0, only the cursor's lines are drawn.

6.11.4 Dettagli

Chart type

A chart can be one of the following types:

- `LV_CHART_TYPE_NONE`: Do not display any data. Can be used to hide chart's data.
- `LV_CHART_TYPE_LINE`: Draw lines between data points. Data points can also be illustrated if their `width`, `height`, `bg_color` and `radius` styles (for `LV_PART_ITEMS`) are set and both `width` and `height` have non-zero values.
- `LV_CHART_TYPE_BAR`: Draw bars.

- *LV_CHART_TYPE_SCATTER*: X/Y chart drawing point's and optionally lines between the points if line-width style values for *LV_PART_ITEMS* is a non-zero value, and the point's Y-value is something other than *LV_CHART_POINT_NONE*. (Drawing of individual points on a SCATTER chart can be suppressed if their Y-values are set to *LV_CHART_POINT_NONE*.)

Charts start their life as LINE charts. You can change a chart's type with *lv_chart_set_type*(chart, LV_CHART_TYPE_...).

Data series

You can add any number of data series to a chart by using

lv_chart_add_series(chart, color, axis).

This allocates (and returns a pointer to) an `lv_chart_series_t` structure which remembers the `color` and `axis` you specified, and comes pre-allocated with an array of `chart->point_cnt` `int32_t` Y-values, all set to *LV_CHART_POINT_NONE*. (A SCATTER chart also comes with a pre-allocated array of the same number of X-values.)

`axis` specifies which axis is used to scale its values, and may be one of the following:

- *LV_CHART_AXIS_PRIMARY_Y*: Left axis
- *LV_CHART_AXIS_SECONDARY_Y*: Right axis
- *LV_CHART_AXIS_PRIMARY_X*: Bottom axis
- *LV_CHART_AXIS_SECONDARY_X*: Top axis

When adding a data series to a SCATTER chart, bit-wise OR your selected Y axis (primary or secondary) with one of the X-axis values.

If you wish to have the chart use your own Y-value array instead of the one provided, you can do so with

lv_chart_set_series_ext_y_array(chart, series, value_array).

You are responsible for ensuring the array you provide contains at least `chart->point_cnt` elements in it.

`value_array` should look like this: `int32_t * value_array[num_points]`. Only the array's pointer is saved in the series so its contents need to remain available for the life of the series, i.e. the array needs to be global, static or dynamically allocated.

> **Nota**
>
> Call *lv_chart_refresh*(chart) when a chart's data has changed to signal that the chart should be re-rendered next time a display refresh occurs. You do not need to do this if you are using the provided value array(s) and setting values with `lv_chart_set_...value_...()` functions. See below for more information about these functions.

A pointer to the Y-value array of a series can be obtained with *lv_chart_get_series_y_array*(chart, series). This is true whether you are using the provided Y-value array or provided your own.

For SCATTER-type charts,

- *lv_chart_set_series_ext_x_array*(chart, series, value_array) and
- *lv_chart_get_series_x_array*(chart, series)

can be used as well.

Modifying data

You have several options to set the Y-values for a data series:

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

2. Use `lv_chart_set_series_value_by_id(chart, series, id, value)` where `id` is the zero-based index of the point you wish to update.
3. Use `lv_chart_set_next_value(chart, series, value)`. (See *Update modes* below.)
4. Set all points to a single Y-value with `lv_chart_set_all_values(chart, series, value)`.

Use `LV_CHART_POINT_NONE` as value to make the library skip drawing that point, column, or scatter-point.

For SCATTER-type charts,

- `lv_chart_set_series_value_by_id2(chart, series, id, x_value, y_value)` and
- `lv_chart_set_next_value2(chart, series, x_value, y_value)`

can be used as well.

Update modes

`lv_chart_set_next_value()` can behave in two ways depending on *update mode*:

- `LV_CHART_UPDATE_MODE_SHIFT`: Shift old data to the left and add the new one to the right.
- `LV_CHART_UPDATE_MODE_CIRCULAR`: Add the new data in circular fashion, like an ECG diagram.

The update mode can be changed with `lv_chart_set_update_mode(chart, LV_CHART_UPDATE_MODE_...)`.

Number of points

The number of points in the series can be modified by `lv_chart_set_point_count(chart, point_num)`. The default value is 10. Note: this affects the number of points processed when an external value array is assigned to a series, so you also need to be sure any external array so provided contains at least `point_num` elements.

Handling large numbers of points

On LINE charts, if the number of points is greater than the pixels horizontally, the Chart will draw only vertical lines to make the drawing of large amount of data effective. If there are, let's say, 10 points to a pixel, LVGL searches the smallest and the largest value and draws a vertical lines between them to ensure no peaks are missed.

Vertical range

You can specify the minimum and maximum values in Y-direction with `lv_chart_set_axis_range(chart, axis, min, max)`. `axis` can be `LV_CHART_AXIS_PRIMARY_Y` (left Y axis) or `LV_CHART_AXIS_SECONDARY_Y` (right Y axis).

The value of the points will be scaled proportionally. The default range is: 0..100.

Division lines

The number of horizontal and vertical division lines can be modified by `lv_chart_set_div_line_count(chart, hdiv_num, vdiv_num)`. The default settings are 3 horizontal and 5 vertical division lines. If there is a visible border on a side and no padding on that side, the division line would be drawn on top of the border and in this case it is not drawn so as not to hide the chart border.

Override default start point for series

If you want a plot to start from a point other than the default which is `point[0]` of the series, you can set an alternative index with the function `lv_chart_set_x_start_point(chart, series, id)` where `id` is the new zero-based index position to start plotting from.

Note that `LV_CHART_UPDATE_MODE_SHIFT` also changes the `start_point`.

Tick marks and labels

With the help of *Scale*, vertical and horizontal scales can be added in a very flexible way. See the *examples 2* below to learn more.

Zoom

To zoom the chart all you need to do is wrap it in a parent container and set the chart's width and/or height to a larger value. Doing this will cause the the chart to be scrollable in its parent --- the parent container provides the scrollable "view window".

Cursor

A new cursor is initially given position `LV_CHART_POINT_NONE` which causes it to be hidden. To show the cursor, its location must be set by you programmatically using one of the functions below.

You can hide a cursor without removing it from the chart by using `lv_chart_set_cursor_point()` by passing `LV_CHART_POINT_NONE` as the point id.

A cursor can be added with `lv_chart_cursor_t * c1 = lv_chart_add_cursor(chart, color, dir);`. The possible values of `dir` are the enumeration values of `lv_dir_t`: `LV_DIR_NONE/RIGHT/UP/LEFT/DOWN/HOR/VER/ALL` or their bit-wise OR-ed values to tell the chart which direction(s) to draw its lines.

`lv_chart_set_cursor_pos(chart, cursor, &point)` sets the position of the cursor to an arbitrary point on the chart. `&point` is a pointer to an `lv_point_t` variable. E.g. `lv_point_t point = {10, 20}`. If the chart is scrolled, the cursor moves with it.

`lv_chart_get_point_pos_by_id(chart, series, id, &point_out)` gets the coordinate of a given point on the chart. This is useful to place the cursor on that data point.

`lv_chart_set_cursor_point(chart, cursor, series, point_id)` places the cursor on the specified data point on the chart. If the point's position changes (via a new value or via scrolling), the cursor moves with the point. See an example of using this function *here*.

6.11.5 Eventi

- `LV_EVENT_VALUE_CHANGED` Sent when a new point on the chart is pressed. `lv_chart_get_pressed_point(chart)` returns the zero-based index of the pressed point.

> **Further Reading**
>
> Learn more about *Base-Widget Events* emitted by all Widgets.
>
> Learn more about *Eventi*.

6.11.6 Keys

No *Keys* are processed by Chart Widgets.

> **Further Reading**
>
> Learn more about *Keys*.

6.11.7 Esempio

Grafico a linee

```
#include "../../lv_examples.h"
#if LV_USE_CHART && LV_BUILD_EXAMPLES

/**
 * A very basic line chart
 */
void lv_example_chart_1(void)
{
    /*Create a chart*/
    lv_obj_t * chart;
    chart = lv_chart_create(lv_screen_active());
    lv_obj_set_size(chart, 200, 150);
    lv_obj_center(chart);
    lv_chart_set_type(chart, LV_CHART_TYPE_LINE);   /*Show lines and␣
↪points too*/

    /*Add two data series*/
    lv_chart_series_t * ser1 = lv_chart_add_series(chart, lv_palette_
↪main(LV_PALETTE_GREEN), LV_CHART_AXIS_PRIMARY_Y);
    lv_chart_series_t * ser2 = lv_chart_add_series(chart, lv_palette_
↪main(LV_PALETTE_RED), LV_CHART_AXIS_SECONDARY_Y);
    int32_t * ser2_y_points = lv_chart_get_series_y_array(chart, ser2);

    uint32_t i;
    for(i = 0; i < 10; i++) {
        /*Set the next points on 'ser1'*/
        lv_chart_set_next_value(chart, ser1, (int32_t)lv_rand(10, 50));

        /*Directly set points on 'ser2'*/
        ser2_y_points[i] = (int32_t)lv_rand(50, 90);
    }

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

#endif
```

Axis ticks and labels with scrolling

```
#include "../../lv_examples.h"
#if LV_USE_CHART && LV_BUILD_EXAMPLES

/**
 * Use lv_scale to add ticks to a scrollable chart
 */
void lv_example_chart_2(void)
{
    /*Create a container*/
    lv_obj_t * main_cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(main_cont, 200, 150);
    lv_obj_center(main_cont);

    /*Create a transparent wrapper for the chart and the scale.
     *Set a large width, to make it scrollable on the main container*/
    lv_obj_t * wrapper = lv_obj_create(main_cont);
    lv_obj_remove_style_all(wrapper);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_size(wrapper, lv_pct(300), lv_pct(100));
    lv_obj_set_flex_flow(wrapper, LV_FLEX_FLOW_COLUMN);

    /*Create a chart on the wrapper
     *Set it's width to 100% to fill the large wrapper*/
    lv_obj_t * chart = lv_chart_create(wrapper);
    lv_obj_set_width(chart, lv_pct(100));
    lv_obj_set_flex_grow(chart, 1);
    lv_chart_set_type(chart, LV_CHART_TYPE_BAR);
    lv_chart_set_axis_range(chart, LV_CHART_AXIS_PRIMARY_Y, 0, 100);
    lv_chart_set_axis_range(chart, LV_CHART_AXIS_SECONDARY_Y, 0, 400);
    lv_chart_set_point_count(chart, 12);
    lv_obj_set_style_radius(chart, 0, 0);

    /*Create a scale also with 100% width*/
    lv_obj_t * scale_bottom = lv_scale_create(wrapper);
    lv_scale_set_mode(scale_bottom, LV_SCALE_MODE_HORIZONTAL_BOTTOM);
    lv_obj_set_size(scale_bottom, lv_pct(100), 25);
    lv_scale_set_total_tick_count(scale_bottom, 12);
    lv_scale_set_major_tick_every(scale_bottom, 1);
    lv_obj_set_style_pad_hor(scale_bottom, lv_chart_get_first_point_center_
↪offset(chart), 0);

    static const char * month[] = {"Jan", "Febr", "March", "Apr", "May",
↪"Jun", "July", "Aug", "Sept", "Oct", "Nov", "Dec", NULL};
    lv_scale_set_text_src(scale_bottom, month);

    /*Add two data series*/
    lv_chart_series_t * ser1 = lv_chart_add_series(chart, lv_palette_
↪lighten(LV_PALETTE_GREEN, 2), LV_CHART_AXIS_PRIMARY_Y);
    lv_chart_series_t * ser2 = lv_chart_add_series(chart, lv_palette_
↪darken(LV_PALETTE_GREEN, 2), LV_CHART_AXIS_PRIMARY_Y);

    /*Set the next points on 'ser1'*/
    uint32_t i;
    for(i = 0; i < 12; i++) {
        lv_chart_set_next_value(chart, ser1, (int32_t)lv_rand(10, 60));
        lv_chart_set_next_value(chart, ser2, (int32_t)lv_rand(50, 90));
    }
    lv_chart_refresh(chart); /*Required after direct set*/
}

#endif
```

Mostrare il valore dei punti premuti

```
#include "../../lv_examples.h"
#if LV_USE_CHART && LV_BUILD_EXAMPLES

static void event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * chart = lv_event_get_target_obj(e);

    if(code == LV_EVENT_VALUE_CHANGED) {
        lv_obj_invalidate(chart);
```

(continues on next page)

(continua dalla pagina precedente)

```
    }
    if(code == LV_EVENT_REFR_EXT_DRAW_SIZE) {
        int32_t * s = (int32_t *)lv_event_get_param(e);
        *s = LV_MAX(*s, 20);
    }
    else if(code == LV_EVENT_DRAW_POST_END) {
        uint32_t id = lv_chart_get_pressed_point(chart);
        if(id == LV_CHART_POINT_NONE) return;

        LV_LOG_USER("Selected point %d", (int)id);

        lv_chart_series_t * ser = lv_chart_get_series_next(chart, NULL);
        while(ser) {
            lv_point_t p;
            lv_chart_get_point_pos_by_id(chart, ser, id, &p);

            int32_t * y_array = lv_chart_get_series_y_array(chart, ser);
            int32_t value = y_array[id];

            /*Draw a rectangle above the clicked point*/
            lv_layer_t * layer = lv_event_get_layer(e);
            lv_draw_rect_dsc_t draw_rect_dsc;
            lv_draw_rect_dsc_init(&draw_rect_dsc);
            draw_rect_dsc.bg_color = lv_color_black();
            draw_rect_dsc.bg_opa = LV_OPA_50;
            draw_rect_dsc.radius = 3;

            lv_area_t chart_obj_coords;
            lv_obj_get_coords(chart, &chart_obj_coords);
            lv_area_t rect_area;
            rect_area.x1 = chart_obj_coords.x1 + p.x - 20;
            rect_area.x2 = chart_obj_coords.x1 + p.x + 20;
            rect_area.y1 = chart_obj_coords.y1 + p.y - 30;
            rect_area.y2 = chart_obj_coords.y1 + p.y - 10;
            lv_draw_rect(layer, &draw_rect_dsc, &rect_area);

            /*Draw the value as label to the center of the rectangle*/
            char buf[16];
            lv_snprintf(buf, sizeof(buf), LV_SYMBOL_DUMMY"$%d", value);

            lv_draw_label_dsc_t draw_label_dsc;
            lv_draw_label_dsc_init(&draw_label_dsc);
            draw_label_dsc.color = lv_color_white();
            draw_label_dsc.text = buf;
            draw_label_dsc.text_local = 1;
            draw_label_dsc.align = LV_TEXT_ALIGN_CENTER;
            lv_area_t label_area = rect_area;
            lv_area_set_height(&label_area, lv_font_get_line_height(draw_
↪label_dsc.font));
            lv_area_align(&rect_area, &label_area, LV_ALIGN_CENTER, 0, 0);
            lv_draw_label(layer, &draw_label_dsc, &label_area);

            ser = lv_chart_get_series_next(chart, ser);
        }
    }
    else if(code == LV_EVENT_RELEASED) {
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_obj_invalidate(chart);
    }
}

/**
 * Show the value of the pressed points
 */
void lv_example_chart_3(void)
{
    /*Create a chart*/
    lv_obj_t * chart;
    chart = lv_chart_create(lv_screen_active());
    lv_obj_set_size(chart, 200, 150);
    lv_obj_center(chart);

    lv_obj_add_event_cb(chart, event_cb, LV_EVENT_ALL, NULL);
    lv_obj_refresh_ext_draw_size(chart);

    /*Zoom in a little in X*/
    //    lv_chart_set_scale_x(chart, 800);

    /*Add two data series*/
    lv_chart_series_t * ser1 = lv_chart_add_series(chart, lv_palette_
↪main(LV_PALETTE_RED), LV_CHART_AXIS_PRIMARY_Y);
    lv_chart_series_t * ser2 = lv_chart_add_series(chart, lv_palette_
↪main(LV_PALETTE_GREEN), LV_CHART_AXIS_PRIMARY_Y);
    uint32_t i;
    for(i = 0; i < 10; i++) {
        lv_chart_set_next_value(chart, ser1, (int32_t)lv_rand(60, 90));
        lv_chart_set_next_value(chart, ser2, (int32_t)lv_rand(10, 40));
    }
}

#endif
```

Ricolorare le barre in base al loro valore

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

#if LV_USE_CHART && LV_DRAW_SW_COMPLEX && LV_BUILD_EXAMPLES

static void draw_event_cb(lv_event_t * e)
{
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);

    if(base_dsc->part != LV_PART_ITEMS) {
        return;
    }

    lv_draw_fill_dsc_t * fill_dsc = lv_draw_task_get_fill_dsc(draw_task);
    if(fill_dsc) {
        lv_obj_t * chart = lv_event_get_target_obj(e);
        int32_t * y_array = lv_chart_get_series_y_array(chart, lv_chart_
↪get_series_next(chart, NULL));
```

(continues on next page)

(continua dalla pagina precedente)

```
        int32_t v = y_array[base_dsc->id2];

        uint8_t ratio = (uint8_t)(v * 255 / 100);
        fill_dsc->color = lv_color_mix(lv_palette_main(LV_PALETTE_GREEN),␣
↪lv_palette_main(LV_PALETTE_RED), ratio);
    }
}

/**
 * Recolor the bars of a chart based on their value
 */
void lv_example_chart_4(void)
{
    /*Create a chart1*/
    lv_obj_t * chart = lv_chart_create(lv_screen_active());
    lv_chart_set_type(chart, LV_CHART_TYPE_BAR);
    lv_chart_set_point_count(chart, 24);
    lv_obj_set_style_pad_column(chart, 2, 0);
    lv_obj_set_size(chart, 260, 160);
    lv_obj_center(chart);

    lv_chart_series_t * ser = lv_chart_add_series(chart, lv_color_
↪hex(0xff0000), LV_CHART_AXIS_PRIMARY_Y);
    lv_obj_add_event_cb(chart, draw_event_cb, LV_EVENT_DRAW_TASK_ADDED,␣
↪NULL);
    lv_obj_add_flag(chart, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);

    uint32_t i;
    for(i = 0; i < 24; i++) {
        lv_chart_set_next_value(chart, ser, (int32_t)lv_rand(10, 90));
    }
}

#endif
```

Grafico a linee sfumate con linee di divisione personalizzate

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

#if LV_USE_CHART && LV_DRAW_SW_COMPLEX && LV_BUILD_EXAMPLES
#include "../../../lvgl_private.h"

static void hook_division_lines(lv_event_t * e);
static void add_faded_area(lv_event_t * e);
static void draw_event_cb(lv_event_t * e);

/**
 * Add a faded area effect to the line chart and make some division lines␣
↪ticker
 */
void lv_example_chart_5(void)
{
    /*Create a chart*/
    lv_obj_t * chart = lv_chart_create(lv_screen_active());
    lv_chart_set_type(chart, LV_CHART_TYPE_LINE);   /*Show lines and␣
↪points too*/
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_size(chart, 200, 150);
    lv_obj_set_style_pad_all(chart, 0, 0);
    lv_obj_set_style_radius(chart, 0, 0);
    lv_obj_center(chart);

    lv_chart_set_div_line_count(chart, 5, 7);

    lv_obj_add_event_cb(chart, draw_event_cb, LV_EVENT_DRAW_TASK_ADDED,␣
↪NULL);
    lv_obj_add_flag(chart, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);

    lv_chart_series_t * ser = lv_chart_add_series(chart, lv_palette_
↪main(LV_PALETTE_RED), LV_CHART_AXIS_PRIMARY_Y);

    uint32_t i;
    for(i = 0; i < 10; i++) {
        lv_chart_set_next_value(chart, ser, (int32_t)lv_rand(10, 80));
    }
}

static void draw_event_cb(lv_event_t * e)
{
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);

    if(base_dsc->part == LV_PART_ITEMS && lv_draw_task_get_type(draw_task)␣
↪== LV_DRAW_TASK_TYPE_LINE) {
        add_faded_area(e);

    }
    /*Hook the division lines too*/
    if(base_dsc->part == LV_PART_MAIN && lv_draw_task_get_type(draw_task)␣
↪== LV_DRAW_TASK_TYPE_LINE) {
        hook_division_lines(e);
    }
}

static void add_faded_area(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    lv_area_t coords;
    lv_obj_get_coords(obj, &coords);

    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);

    const lv_chart_series_t * ser = lv_chart_get_series_next(obj, NULL);
    lv_color_t ser_color = lv_chart_get_series_color(obj, ser);

    /*Draw a triangle below the line witch some opacity gradient*/
    lv_draw_line_dsc_t * draw_line_dsc = (lv_draw_line_dsc_t *)lv_draw_
↪task_get_draw_dsc(draw_task);
    lv_draw_triangle_dsc_t tri_dsc;
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_draw_triangle_dsc_init(&tri_dsc);
    tri_dsc.p[0].x = draw_line_dsc->p1.x;
    tri_dsc.p[0].y = draw_line_dsc->p1.y;
    tri_dsc.p[1].x = draw_line_dsc->p2.x;
    tri_dsc.p[1].y = draw_line_dsc->p2.y;
    tri_dsc.p[2].x = draw_line_dsc->p1.y < draw_line_dsc->p2.y ? draw_line_
↪dsc->p1.x : draw_line_dsc->p2.x;
    tri_dsc.p[2].y = LV_MAX(draw_line_dsc->p1.y, draw_line_dsc->p2.y);
    tri_dsc.grad.dir = LV_GRAD_DIR_VER;

    int32_t full_h = lv_obj_get_height(obj);
    int32_t fract_uppter = (int32_t)(LV_MIN(draw_line_dsc->p1.y, draw_line_
↪dsc->p2.y) - coords.y1) * 255 / full_h;
    int32_t fract_lower = (int32_t)(LV_MAX(draw_line_dsc->p1.y, draw_line_
↪dsc->p2.y) - coords.y1) * 255 / full_h;
    tri_dsc.grad.stops[0].color = ser_color;
    tri_dsc.grad.stops[0].opa = (lv_opa_t)(255 - fract_uppter);
    tri_dsc.grad.stops[0].opa = 255 - fract_uppter;
    tri_dsc.grad.stops[0].frac = 0;
    tri_dsc.grad.stops[1].color = ser_color;
    tri_dsc.grad.stops[1].opa = (lv_opa_t)(255 - fract_lower);
    tri_dsc.grad.stops[1].frac = 255;

    lv_draw_triangle(base_dsc->layer, &tri_dsc);

    /*Draw rectangle below the triangle*/
    lv_draw_rect_dsc_t rect_dsc;
    lv_draw_rect_dsc_init(&rect_dsc);
    rect_dsc.bg_grad.dir = LV_GRAD_DIR_VER;
    rect_dsc.bg_grad.stops[0].color = ser_color;
    rect_dsc.bg_grad.stops[0].frac = 0;
    rect_dsc.bg_grad.stops[0].opa = (lv_opa_t)(255 - fract_lower);
    rect_dsc.bg_grad.stops[1].color = ser_color;
    rect_dsc.bg_grad.stops[1].frac = 255;
    rect_dsc.bg_grad.stops[1].opa = 0;

    lv_area_t rect_area;
    rect_area.x1 = (int32_t)draw_line_dsc->p1.x;
    rect_area.x2 = (int32_t)draw_line_dsc->p2.x - 1;
    rect_area.y1 = (int32_t)LV_MAX(draw_line_dsc->p1.y, draw_line_dsc->p2.
↪y) - 1;
    rect_area.y2 = (int32_t)coords.y2;
    lv_draw_rect(base_dsc->layer, &rect_dsc, &rect_area);
}

static void hook_division_lines(lv_event_t * e)
{
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);
    lv_draw_line_dsc_t * line_dsc = (lv_draw_line_dsc_t *)lv_draw_task_get_
↪draw_dsc(draw_task);

    /*Vertical line*/
    if(line_dsc->p1.x == line_dsc->p2.x) {
        line_dsc->color  = lv_palette_lighten(LV_PALETTE_GREY, 1);
```

(continues on next page)

(continua dalla pagina precedente)

```
        if(base_dsc->id1 == 3) {
            line_dsc->width = 2;
            line_dsc->dash_gap = 0;
            line_dsc->dash_width = 0;
        }
        else {
            line_dsc->width = 1;
            line_dsc->dash_gap = 6;
            line_dsc->dash_width = 6;
        }
    }
    /*Horizontal line*/
    else {
        if(base_dsc->id1 == 2) {
            line_dsc->width  = 2;
            line_dsc->dash_gap  = 0;
            line_dsc->dash_width  = 0;
        }
        else {
            line_dsc->width = 2;
            line_dsc->dash_gap  = 6;
            line_dsc->dash_width  = 6;
        }

        if(base_dsc->id1 == 1  || base_dsc->id1 == 3) {
            line_dsc->color  = lv_palette_main(LV_PALETTE_GREEN);
        }
        else {
            line_dsc->color  = lv_palette_lighten(LV_PALETTE_GREY, 1);
        }
    }
}

#endif
```

Mostrare il cursore sul punto cliccato

```
#include "../../lv_examples.h"
#if LV_USE_CHART && LV_BUILD_EXAMPLES

static lv_obj_t * chart;
static lv_chart_series_t * ser;
static lv_chart_cursor_t * cursor;

static void value_changed_event_cb(lv_event_t * e)
{
    uint32_t last_id;
    lv_obj_t * obj = lv_event_get_target_obj(e);

    last_id = lv_chart_get_pressed_point(obj);
    if(last_id != LV_CHART_POINT_NONE) {
        lv_chart_set_cursor_point(obj, cursor, NULL, last_id);
    }
}

/**
```

(continues on next page)

(continua dalla pagina precedente)

```
 * Show cursor on the clicked point
 */
void lv_example_chart_6(void)
{
    chart = lv_chart_create(lv_screen_active());
    lv_obj_set_size(chart, 200, 150);
    lv_obj_align(chart, LV_ALIGN_CENTER, 0, -10);

    //    lv_chart_set_axis_tick(chart, LV_CHART_AXIS_PRIMARY_Y, 10, 5, 6,␣
↪5, true, 40);
    //    lv_chart_set_axis_tick(chart, LV_CHART_AXIS_PRIMARY_X, 10, 5, 10,
↪ 1, true, 30);

    lv_obj_add_event_cb(chart, value_changed_event_cb, LV_EVENT_VALUE_
↪CHANGED, NULL);
    lv_obj_refresh_ext_draw_size(chart);

    cursor = lv_chart_add_cursor(chart, lv_palette_main(LV_PALETTE_BLUE),␣
↪(lv_dir_t)(LV_DIR_LEFT | LV_DIR_BOTTOM));

    ser = lv_chart_add_series(chart, lv_palette_main(LV_PALETTE_RED), LV_
↪CHART_AXIS_PRIMARY_Y);
    uint32_t i;
    for(i = 0; i < 10; i++) {
        lv_chart_set_next_value(chart, ser, (int32_t)lv_rand(10, 90));
    }

    //    lv_chart_set_scale_x(chart, 500);

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Click on a point");
    lv_obj_align_to(label, chart, LV_ALIGN_OUT_TOP_MID, 0, -5);
}

#endif
```

Grafico a dispersione

```
#include "../../lv_examples.h"
#if LV_USE_CHART && LV_BUILD_EXAMPLES

static void draw_event_cb(lv_event_t * e)
{
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);
    if(base_dsc->part == LV_PART_INDICATOR) {
        lv_obj_t * obj = lv_event_get_target_obj(e);
        lv_chart_series_t * ser = lv_chart_get_series_next(obj, NULL);
        lv_draw_rect_dsc_t * rect_draw_dsc = (lv_draw_rect_dsc_t *)lv_draw_
↪task_get_draw_dsc(draw_task);
        uint32_t cnt = lv_chart_get_point_count(obj);

        /*Make older value more transparent*/
        rect_draw_dsc->bg_opa = (lv_opa_t)((LV_OPA_COVER * base_dsc->id2) /
↪ (cnt - 1));
```

(continues on next page)

(continua dalla pagina precedente)

```
        /*Make smaller values blue, higher values red*/
        int32_t * x_array = lv_chart_get_series_x_array(obj, ser);
        int32_t * y_array = lv_chart_get_series_y_array(obj, ser);
        /*dsc->id is the tells drawing order, but we need the ID of the␣
↪point being drawn.*/
        uint32_t start_point = lv_chart_get_x_start_point(obj, ser);
        uint32_t p_act = (start_point + base_dsc->id2) % cnt; /*Consider␣
↪start point to get the index of the array*/
        lv_opa_t x_opa = (lv_opa_t)((x_array[p_act] * LV_OPA_50) / 200);
        lv_opa_t y_opa = (lv_opa_t)((y_array[p_act] * LV_OPA_50) / 1000);

        rect_draw_dsc->bg_color = lv_color_mix(lv_palette_main(LV_PALETTE_
↪RED),
                                               lv_palette_main(LV_PALETTE_
↪BLUE),
                                               x_opa + y_opa);
    }
}

static void add_data(lv_timer_t * timer)
{
    lv_obj_t * chart = (lv_obj_t *)lv_timer_get_user_data(timer);
    lv_chart_set_next_value2(chart, lv_chart_get_series_next(chart, NULL),␣
↪(int32_t)lv_rand(0, 200),
                             (int32_t)lv_rand(0, 1000));
}

/**
 * A scatter chart
 */
void lv_example_chart_7(void)
{
    lv_obj_t * chart = lv_chart_create(lv_screen_active());
    lv_obj_set_size(chart, 200, 150);
    lv_obj_align(chart, LV_ALIGN_CENTER, 0, 0);
    lv_obj_add_event_cb(chart, draw_event_cb, LV_EVENT_DRAW_TASK_ADDED,␣
↪NULL);
    lv_obj_add_flag(chart, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);
    lv_obj_set_style_line_width(chart, 0, LV_PART_ITEMS);   /*Remove the␣
↪lines*/

    lv_chart_set_type(chart, LV_CHART_TYPE_SCATTER);

    lv_chart_set_axis_range(chart, LV_CHART_AXIS_PRIMARY_X, 0, 200);
    lv_chart_set_axis_range(chart, LV_CHART_AXIS_PRIMARY_Y, 0, 1000);

    lv_chart_set_point_count(chart, 50);

    lv_chart_series_t * ser = lv_chart_add_series(chart, lv_palette_
↪main(LV_PALETTE_RED), LV_CHART_AXIS_PRIMARY_Y);
    uint32_t i;
    for(i = 0; i < 50; i++) {
        lv_chart_set_next_value2(chart, ser, (int32_t)lv_rand(0, 200),␣
↪(int32_t)lv_rand(0, 1000));
    }
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_timer_create(add_data, 100, chart);
}

#endif
```

Grafico a linee circolari con spazio

```
#include "../../lv_examples.h"
#if LV_USE_CHART && LV_DRAW_SW_COMPLEX && LV_BUILD_EXAMPLES

static void add_data(lv_timer_t * t)
{
    lv_obj_t * chart = (lv_obj_t *)lv_timer_get_user_data(t);
    lv_chart_series_t * ser = lv_chart_get_series_next(chart, NULL);

    lv_chart_set_next_value(chart, ser, (int32_t)lv_rand(10, 90));

    uint32_t p = lv_chart_get_point_count(chart);
    uint32_t s = lv_chart_get_x_start_point(chart, ser);
    int32_t * a = lv_chart_get_series_y_array(chart, ser);

    a[(s + 1) % p] = LV_CHART_POINT_NONE;
    a[(s + 2) % p] = LV_CHART_POINT_NONE;
    a[(s + 2) % p] = LV_CHART_POINT_NONE;

    lv_chart_refresh(chart);
}

/**
 * Circular line chart with gap
 */
void lv_example_chart_8(void)
{
    /*Create a stacked_area_chart.obj*/
    lv_obj_t * chart = lv_chart_create(lv_screen_active());
    lv_chart_set_update_mode(chart, LV_CHART_UPDATE_MODE_CIRCULAR);
    lv_obj_set_style_size(chart, 0, 0, LV_PART_INDICATOR);
    lv_obj_set_size(chart, 280, 150);
    lv_obj_center(chart);

    lv_chart_set_point_count(chart, 80);
    lv_chart_series_t * ser = lv_chart_add_series(chart, lv_palette_
↪main(LV_PALETTE_RED), LV_CHART_AXIS_PRIMARY_Y);
    /*Prefill with data*/
    uint32_t i;
    for(i = 0; i < 80; i++) {
        lv_chart_set_next_value(chart, ser, (int32_t)lv_rand(10, 90));
    }

    lv_timer_create(add_data, 300, chart);

}

#endif
```

6.11.8 API

lv_api_map_v9_1.h

lv_chart.h

lv_chart_private.h

6.12 Checkbox (lv_checkbox)

6.12.1 Panoramica

The Checkbox Widget is created from a "tick box" and a label. When the Checkbox is clicked the tick box is toggled.

6.12.2 Parts and Styles

- `LV_PART_MAIN` Background of Checkbox and it uses the text and the *typical background style properties*. `pad_column` adjusts spacing between tickbox and label
- `LV_PART_INDICATOR` The "tick box" is a square that uses the *typical background style properties*. By default, its size is equal to the height of the main part's font. Padding properties make the tick box larger in the respective directions.

The Checkbox is added to the default group (if one is set).

6.12.3 Utilizzo

Text

The text can be modified with `lv_checkbox_set_text(cb, "New text")` and will be dynamically allocated.

To set static text, use `lv_checkbox_set_text_static(cb, txt)`. This way, only a pointer to `txt` will be stored. The provided text buffer must remain available for the lifetime of the Checkbox.

Check, uncheck, disable

You can programmatically check, un-check, and disable the Checkbox by using the common state add/clear function:

```
lv_obj_add_state(cb, LV_STATE_CHECKED);    /* Make Checkbox checked */
lv_obj_remove_state(cb, LV_STATE_CHECKED); /* Make Checkbox unchecked */
lv_obj_add_state(cb, LV_STATE_CHECKED);    /* Make Checkbox checked */
lv_obj_add_state(cb, LV_STATE_DISABLED);   /* Make Checkbox disabled */
```

To find out whether the Checkbox is checked use `lv_obj_has_state(cb, LV_STATE_CHECKED)`.

6.12.4 Eventi

- `LV_EVENT_VALUE_CHANGED` Sent when Checkbox is toggled.

> **Further Reading**
>
> Learn more about *Base-Widget Events* emitted by all Widgets.
>
> Learn more about *Eventi*.

6.12.5 Keys

The following *Keys* are processed by Checkbox:

- `LV_KEY_RIGHT/UP` Go to CHECKED state if Checkbox is enabled

- `LV_KEY_LEFT/DOWN` Go to non-CHECKED state if Checkbox is enabled
- *`LV_KEY_ENTER`* Clicks the Checkbox and toggles its value.

Note that, as usual, the state of *`LV_KEY_ENTER`* is translated to `LV_EVENT_PRESSED/PRESSING/RELEASED` etc.

Further Reading

Learn more about *Keys*.

6.12.6 Esempio

Checkbox Semplici

```
#include "../../lv_examples.h"
#if LV_USE_CHECKBOX && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        LV_UNUSED(obj);
        const char * txt = lv_checkbox_get_text(obj);
        const char * state = lv_obj_get_state(obj) & LV_STATE_CHECKED ?
↪"Checked" : "Unchecked";
        LV_UNUSED(txt);
        LV_UNUSED(state);
        LV_LOG_USER("%s: %s", txt, state);
    }
}

void lv_example_checkbox_1(void)
{
    lv_obj_set_flex_flow(lv_screen_active(), LV_FLEX_FLOW_COLUMN);
    lv_obj_set_flex_align(lv_screen_active(), LV_FLEX_ALIGN_CENTER, LV_
↪FLEX_ALIGN_START, LV_FLEX_ALIGN_CENTER);

    lv_obj_t * cb;
    cb = lv_checkbox_create(lv_screen_active());
    lv_checkbox_set_text(cb, "Apple");
    lv_obj_add_event_cb(cb, event_handler, LV_EVENT_ALL, NULL);

    cb = lv_checkbox_create(lv_screen_active());
    lv_checkbox_set_text(cb, "Banana");
    lv_obj_add_state(cb, LV_STATE_CHECKED);
    lv_obj_add_event_cb(cb, event_handler, LV_EVENT_ALL, NULL);

    cb = lv_checkbox_create(lv_screen_active());
    lv_checkbox_set_text(cb, "Lemon");
    lv_obj_add_state(cb, LV_STATE_DISABLED);
    lv_obj_add_event_cb(cb, event_handler, LV_EVENT_ALL, NULL);

    cb = lv_checkbox_create(lv_screen_active());
    lv_obj_add_state(cb, LV_STATE_CHECKED);
    lv_obj_add_state(cb, LV_STATE_DISABLED);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_checkbox_set_text(cb, "Melon\nand a new line");
    lv_obj_add_event_cb(cb, event_handler, LV_EVENT_ALL, NULL);

    lv_obj_update_layout(cb);
}

#endif
```

Checkboxes come come radio button

```
#include "../../lv_examples.h"
#if LV_USE_CHECKBOX && LV_BUILD_EXAMPLES

static lv_style_t style_radio;
static lv_style_t style_radio_chk;
static int32_t active_index_1 = 0;
static int32_t active_index_2 = 0;

static void radio_event_handler(lv_event_t * e)
{
    int32_t * active_id = (int32_t *)lv_event_get_user_data(e);
    lv_obj_t * cont = (lv_obj_t *)lv_event_get_current_target(e);
    lv_obj_t * act_cb = lv_event_get_target_obj(e);
    lv_obj_t * old_cb = lv_obj_get_child(cont, *active_id);

    /*Do nothing if the container was clicked*/
    if(act_cb == cont) return;

    lv_obj_remove_state(old_cb, LV_STATE_CHECKED);   /*Uncheck the␣
↪previous radio button*/
    lv_obj_add_state(act_cb, LV_STATE_CHECKED);     /*Check the current␣
↪radio button*/

    *active_id = lv_obj_get_index(act_cb);

    LV_LOG_USER("Selected radio buttons: %d, %d", (int)active_index_1,␣
↪(int)active_index_2);
}

static void radiobutton_create(lv_obj_t * parent, const char * txt)
{
    lv_obj_t * obj = lv_checkbox_create(parent);
    lv_checkbox_set_text(obj, txt);
    lv_obj_add_flag(obj, LV_OBJ_FLAG_EVENT_BUBBLE);
    lv_obj_add_style(obj, &style_radio, LV_PART_INDICATOR);
    lv_obj_add_style(obj, &style_radio_chk, LV_PART_INDICATOR | LV_STATE_
↪CHECKED);
}

/**
 * Checkboxes as radio buttons
 */
void lv_example_checkbox_2(void)
{
    /* The idea is to enable `LV_OBJ_FLAG_EVENT_BUBBLE` on checkboxes and␣
↪process the
```

(continues on next page)

(continua dalla pagina precedente)

```
     * `LV_EVENT_CLICKED` on the container.
     * A variable is passed as event user data where the index of the␣
↪active
     * radiobutton is saved */

    lv_style_init(&style_radio);
    lv_style_set_radius(&style_radio, LV_RADIUS_CIRCLE);

    lv_style_init(&style_radio_chk);
    lv_style_set_bg_image_src(&style_radio_chk, NULL);

    uint32_t i;
    char buf[32];

    lv_obj_t * cont1 = lv_obj_create(lv_screen_active());
    lv_obj_set_flex_flow(cont1, LV_FLEX_FLOW_COLUMN);
    lv_obj_set_size(cont1, lv_pct(40), lv_pct(80));
    lv_obj_add_event_cb(cont1, radio_event_handler, LV_EVENT_CLICKED, &
↪active_index_1);

    for(i = 0; i < 5; i++) {
        lv_snprintf(buf, sizeof(buf), "A %d", (int)i + 1);
        radiobutton_create(cont1, buf);

    }
    /*Make the first checkbox checked*/
    lv_obj_add_state(lv_obj_get_child(cont1, 0), LV_STATE_CHECKED);

    lv_obj_t * cont2 = lv_obj_create(lv_screen_active());
    lv_obj_set_flex_flow(cont2, LV_FLEX_FLOW_COLUMN);
    lv_obj_set_size(cont2, lv_pct(40), lv_pct(80));
    lv_obj_set_x(cont2, lv_pct(50));
    lv_obj_add_event_cb(cont2, radio_event_handler, LV_EVENT_CLICKED, &
↪active_index_2);

    for(i = 0; i < 3; i++) {
        lv_snprintf(buf, sizeof(buf), "B %d", (int)i + 1);
        radiobutton_create(cont2, buf);
    }

    /*Make the first checkbox checked*/
    lv_obj_add_state(lv_obj_get_child(cont2, 0), LV_STATE_CHECKED);
}

#endif
```

6.12.7 API

lv_checkbox.h

lv_checkbox_private.h

6.13 Drop-Down List (lv_dropdown)

6.13.1 Panoramica

The Drop-Down List allows the user to select a value from a list.

The Drop-Down List is closed by default and displays a single value or predefined text. When activated (by click on the Drop-Down List), a list is created from which the user may select one item. When the user selects a new value, the list is deleted again.

The Drop-down list is added to the default group (if one is set). The Drop-down list is an editable Widget allowing list-item selection via encoder or keyboard navigation as well.

6.13.2 Parts and Styles

The Drop-Down List Widget is built from the elements: "button" and "list" (lightweight versions of the Button and List Widgets).

Pulsante

- `LV_PART_MAIN` Background of button, uses the *typical background* and text style properties for its text.
- `LV_PART_INDICATOR` Typically an arrow symbol that can be an Image or text (e.g. `LV_SYMBOL`).

The button goes to `LV_STATE_CHECKED` when it's opened.

Lista

- `LV_PART_MAIN` The list itself; uses the *typical background style properties*. `max_height` can be used to limit the height of the list.
- `LV_PART_SCROLLBAR` The scrollbar background, border, shadow properties and width (for its own width) and right padding for the spacing on the right.
- `LV_PART_SELECTED` Refers to the currently pressed, checked or pressed+checked option. Also uses the *typical background style properties*.

The list is shown/hidden on open/close. To add styles to it use `lv_dropdown_get_list(dropdown)` to get the list object. Esempio:

```
lv_obj_t * list = lv_dropdown_get_list(dropdown) /* Get list */
lv_obj_add_style(list, &my_style, selector)      /* Add styles to list */
```

Alternatively the theme can be extended with new styles.

6.13.3 Utilizzo

List items

The list items are passed to the Drop-Down List as a newline-separated list in a string as the `options` argument to `lv_dropdown_set_options(dropdown, options)`. Each list item should be separated by `\n`. Example: `"First\nSecond\nThird"`. This string is copied by the Drop-Down List, so its contents do not need to remain available beyond this call.

The `lv_dropdown_add_option(dropdown, "New option", pos)` function inserts a new option at index `pos`.

To save memory the options can be set from a static (const) string as well with `lv_dropdown_set_options_static(dropdown, options)`. In this case the options string's contents must remain available for the life of the Drop-Down List and `lv_dropdown_add_option()` cannot be used.

You can select an option programmatically with `lv_dropdown_set_selected(dropdown, id)`, where `id` is the index of the target option.

Get selected option

To get the *index* of the selected option, use `lv_dropdown_get_selected(dropdown)`.

`lv_dropdown_get_selected_str(dropdown, buf, buf_size)` copies the *name* of the selected option to `buf`.

Direction

The list can be created on any side. The default `LV_DIR_BOTTOM` can be modified using `lv_dropdown_set_dir(dropdown, LV_DIR_LEFT)`.

If the list would be vertically out of the screen, it will be aligned to the edge.

Symbol

A symbol (typically an arrow) can be added to the Drop-Down List with `lv_dropdown_set_symbol(dropdown, LV_SYMBOL_...)`

If the direction of the Drop-Down List is `LV_DIR_LEFT` the symbol will be shown on the left, otherwise on the right.

Show selected

The main part can either show the selected item or static text. If static is set with `lv_dropdown_set_text(dropdown, "Some text")` it will be shown regardless of the selected item. If the text is `NULL` the selected option is displayed on the button.

Programmatically open/close

To programmatically open or close the Drop-Down List use `lv_dropdown_open(dropdown)` or `lv_dropdown_close(dropdown)`.

6.13.4 Eventi

- `LV_EVENT_VALUE_CHANGED` Sent when a new option is selected or the list is opened/closed.
- `LV_EVENT_CANCEL` Sent when list is closed.
- `LV_EVENT_READY` Sent when list is opened.

> **Further Reading**
>
> Learn more about *Base-Widget Events* emitted by all Widgets.
>
> Learn more about *Eventi*.

6.13.5 Keys

- `LV_KEY_RIGHT/DOWN` Select next list item.
- `LV_KEY_LEFT/UP` Select previous list item.
- `LV_KEY_ENTER` Apply selected list item (sends `LV_EVENT_VALUE_CHANGED` event and closes Drop-Down List).

> **Further Reading**
>
> Learn more about *Keys*.

6.13.6 Esempio

Menù a discesa semplice

```
#include "../../lv_examples.h"
#if LV_USE_DROPDOWN && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        char buf[32];
        lv_dropdown_get_selected_str(obj, buf, sizeof(buf));
        LV_LOG_USER("Option: %s", buf);
    }
}

void lv_example_dropdown_1(void)
{

    /*Create a normal drop down list*/
    lv_obj_t * dd = lv_dropdown_create(lv_screen_active());
    lv_dropdown_set_options(dd, "Apple\n"
                                "Banana\n"
                                "Orange\n"
                                "Cherry\n"
                                "Grape\n"
                                "Raspberry\n"
                                "Melon\n"
                                "Orange\n"
                                "Lemon\n"
                                "Nuts");

    lv_obj_align(dd, LV_ALIGN_TOP_MID, 0, 20);
    lv_obj_add_event_cb(dd, event_handler, LV_EVENT_ALL, NULL);
}


#endif
```

Menù a discesa nelle quattro direzioni

```
#include "../../lv_examples.h"
#if LV_USE_DROPDOWN && LV_BUILD_EXAMPLES

/**
 * Create a drop down, up, left and right menus
 */
void lv_example_dropdown_2(void)
{
    static const char * opts = "Apple\n"
                               "Banana\n"
                               "Orange\n"
                               "Melon";

    lv_obj_t * dd;
```

(continues on next page)

(continua dalla pagina precedente)

```
    dd = lv_dropdown_create(lv_screen_active());
    lv_dropdown_set_options_static(dd, opts);
    lv_obj_align(dd, LV_ALIGN_TOP_MID, 0, 10);

    dd = lv_dropdown_create(lv_screen_active());
    lv_dropdown_set_options_static(dd, opts);
    lv_dropdown_set_dir(dd, LV_DIR_BOTTOM);
    lv_dropdown_set_symbol(dd, LV_SYMBOL_UP);
    lv_obj_align(dd, LV_ALIGN_BOTTOM_MID, 0, -10);

    dd = lv_dropdown_create(lv_screen_active());
    lv_dropdown_set_options_static(dd, opts);
    lv_dropdown_set_dir(dd, LV_DIR_RIGHT);
    lv_dropdown_set_symbol(dd, LV_SYMBOL_RIGHT);
    lv_obj_align(dd, LV_ALIGN_LEFT_MID, 10, 0);

    dd = lv_dropdown_create(lv_screen_active());
    lv_dropdown_set_options_static(dd, opts);
    lv_dropdown_set_dir(dd, LV_DIR_LEFT);
    lv_dropdown_set_symbol(dd, LV_SYMBOL_LEFT);
    lv_obj_align(dd, LV_ALIGN_RIGHT_MID, -10, 0);
}

#endif
```

Menù

```
#include "../../lv_examples.h"
#if LV_USE_DROPDOWN && LV_BUILD_EXAMPLES

static void event_cb(lv_event_t * e)
{
    lv_obj_t * dropdown = lv_event_get_target_obj(e);
    char buf[64];
    lv_dropdown_get_selected_str(dropdown, buf, sizeof(buf));
    LV_LOG_USER("'%s' is selected", buf);
}

/**
 * Create a menu from a drop-down list and show some drop-down list␣
 ↪features and styling
 */
void lv_example_dropdown_3(void)
{
    /*Create a drop down list*/
    lv_obj_t * dropdown = lv_dropdown_create(lv_screen_active());
    lv_obj_align(dropdown, LV_ALIGN_TOP_LEFT, 10, 10);
    lv_dropdown_set_options(dropdown, "New project\n"
                            "New file\n"
                            "Save\n"
                            "Save as ...\n"
                            "Open project\n"
                            "Recent projects\n"
                            "Preferences\n"
                            "Exit");
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Set a fixed text to display on the button of the drop-down list*/
    lv_dropdown_set_text(dropdown, "Menu");

    /*Use a custom image as down icon and flip it when the list is opened*/
    LV_IMAGE_DECLARE(img_caret_down);
    lv_dropdown_set_symbol(dropdown, &img_caret_down);
    lv_obj_set_style_transform_rotation(dropdown, 1800, LV_PART_INDICATOR␣
↪| LV_STATE_CHECKED);

    /*In a menu we don't need to show the last clicked item*/
    lv_dropdown_set_selected_highlight(dropdown, false);

    lv_obj_add_event_cb(dropdown, event_cb, LV_EVENT_VALUE_CHANGED, NULL);
}

#endif
```

6.13.7 API

lv_api_map_v8.h

lv_dropdown.h

lv_dropdown_private.h

lv_obj_property_names.h

lv_observer.h

6.14 Image (lv_image)

6.14.1 Panoramica

Images are Widgets that display images from flash (as arrays) or from files. Images can also display symbols (`LV_SYMBOL_...`).

Using the *Image decoder interface*, custom image formats can be supported as well.

6.14.2 Parts and Styles

- *`LV_PART_MAIN`* A background rectangle that uses the *typical background style properties*, and the image itself uses the image style properties.

6.14.3 Utilizzo

Image source

To provide maximum flexibility, the source of the image can be:

- a variable in code (a C array containing the pixels).
- a file stored externally (e.g. on an SD card).
- a *Symbol* as text.

To set the source of an image, use *`lv_image_set_src`*`(img, src)`.

To generate a pixel array from a PNG, JPG or BMP image, use the Online image converter tool and set the converted image as the image source with its pointer with *`lv_image_set_src`*`(img1, &converted_img_var)`. To make the converted image variable accessible from the C file, declare it with `LV_IMAGE_DECLARE(converted_img_var)`.

To use external files, you also need to convert the image files using the online converter tool, but select the binary output format. You also need to use LVGL's file system module and register a driver with some functions for basic file operations. See *File system* to learn more. Then set the translated image as the image source with `lv_image_set_src(img, "S:folder1/my_img.bin")`.

You can also set a symbol as an image source similar to a *Labels*. In this case, the image will be rendered as text according to the *font* specified in the style. It enables the use of light-weight monochrome "characters" instead of real images. You can set a symbol as an image source with `lv_image_set_src(img1, LV_SYMBOL_OK)`.

Label as an image

Images and labels are sometimes used to convey the same thing, such as describing what a button does. In this context, images and labels are somewhat interchangeable: images can display text by using the macro `LV_SYMBOL_DUMMY` (which equates to a 3-byte C string containing a special code) as the prefix of the text. For example, `lv_image_set_src(img, LV_SYMBOL_DUMMY "Some text")`.

Transparency

The internal (pixel array) and external images support 2 transparency handling methods:

- **Alpha byte**: An alpha channel is added to every pixel that contains its opacity, typically a byte. It is the 'A' in the various color formats that contain an alpha channel, such as ARGB8888, ARGB8565, ARGB1555, etc.
- **Indexed transparent color**: a specific index in a color palette serves to signal transparency for each pixel that uses it.

Palette and Alpha index

Besides the *True color* (RGB) color format, the following formats are supported:

- **Indexed**: Image has a color palette, and each pixel is an index into that palette.
- **Alpha indexed**: The values stored at pixel positions are alpha (opacity) values.

These options can be selected in the image converter. Learn more about color formats in the *Color formats* section.

Recolor

A color can be mixed with every pixel of an image with a given intensity. This can be useful to show different states (checked, inactive, pressed, etc.) of an image without storing more versions of the same image. This feature can be enabled in the style by setting `img_recolor_opa` between `LV_OPA_TRANSP` (no recolor, value: 0) and `LV_OPA_COVER` (full recolor, value: 255). The default value is `LV_OPA_TRANSP` causing this feature to be disabled.

The color to mix is set by `img_recolor`.

Offset

With `lv_image_set_offset_x(img, x_ofs)` and `lv_image_set_offset_y(img, y_ofs)`, you can add some offset to the displayed image. Useful if the Widget size is smaller than the image source size. Using the offset parameter a Texture atlas or a "running image" effect can be created by *Animating* the x or y offset.

Transformations

You can zoom images in or out by using `lv_image_set_scale(img, factor)`. Set `factor` to `256` or `LV_SCALE_NONE` to disable zooming. A larger value enlarges the images (e.g. `512` double size), a smaller value shrinks it (e.g. `128` half size). Fractional scaling works using a value that is proportionally larger or smaller, e.g. `281` for 10% enlargement.

`lv_image_set_scale_x(img, factor)` and `lv_image_set_scale_y(img, factor)` can also be used to set the horizontal and vertical scaling independently. They can be different values.

To rotate the image use `lv_image_set_rotation(img, angle_x10)`. The `angle_x10` argument is an `int32_t` containing the angle (in degrees) multiplied by 10. This gives 0.1-degree resolution. Example: 458 means 45.8°.

By default, the pivot point of the rotation is the center of the image. This can be changed with `lv_image_set_pivot(img, pivot_x, pivot_y)` where the coordinates `(0,0)` represent the top left corner.

The quality of the transformation can be adjusted with `lv_image_set_antialias(img, true)`. Enabling anti-aliasing causes the transformations to be of higher quality, but slower.

Transformations require the whole image to be available. Therefore indexed images (`LV_COLOR_FORMAT_I1/2/4/8_...`) and alpha only images cannot be transformed. In other words transformations work only on normal (A)RGB or A8 images stored as a C array, or on images provided by a custom *Image decoder* that returns the whole image.

Note that the real coordinates of image Widgets do not change with a transformation. That is `lv_obj_get_width / height / x / y()` will return the original, non-zoomed coordinates.

IMPORTANT: The transformation of the image is independent of the transformation properties *coming from styles*. The main differences are that pure Image Widget transformations:

- do not transform the children of the Image Widget, and
- the image is transformed directly without creating an intermediate layer (buffer) to snapshot the Widget.

Inner align

By default the image Widget's width and height are `LV_SIZE_CONTENT`, meaning that the Widget will be sized automatically to the size of its image source.

If the Widget's width or height is set to a different value, the value of the `inner_align` property (set using `lv_image_set_inner_align(widget, align)`) governs how the image source is aligned inside the Widget.

`align` can be any of these values:

- *`LV_IMAGE_ALIGN_DEFAULT`*: Meaning top left
- *`LV_IMAGE_ALIGN_TOP_LEFT`*
- *`LV_IMAGE_ALIGN_TOP_MID`*
- *`LV_IMAGE_ALIGN_TOP_RIGHT`*
- *`LV_IMAGE_ALIGN_BOTTOM_LEFT`*
- *`LV_IMAGE_ALIGN_BOTTOM_MID`*
- *`LV_IMAGE_ALIGN_BOTTOM_RIGHT`*
- *`LV_IMAGE_ALIGN_LEFT_MID`*
- *`LV_IMAGE_ALIGN_RIGHT_MID`*
- *`LV_IMAGE_ALIGN_CENTER`*
- *`LV_IMAGE_ALIGN_STRETCH`*
- *`LV_IMAGE_ALIGN_TILE`*
- *`LV_IMAGE_ALIGN_CONTAIN`*
- *`LV_IMAGE_ALIGN_COVER`*

Any `offset` value is applied after the image source is aligned. For example setting an offset of `y=-10` with `align == LV_IMAGE_ALIGN_CENTER` will move the image source up 10 pixels from the center of the Widget.

To automatically scale or tile the image, pass one of these `align` values:

- *`LV_IMAGE_ALIGN_STRETCH`* Set X and Y scale to fill the Widget's area

- *LV_IMAGE_ALIGN_TILE* Tile image to fill Widget's area. Offset is applied to shift the tiling.
- *LV_IMAGE_ALIGN_CONTAIN* The image keeps its aspect ratio, but is resized to the maximum size that fits within the Widget's area.
- *LV_IMAGE_ALIGN_COVER* The image keeps its aspect ratio and fills the Widget's area.

6.14.4 Eventi

No special events are sent by Image Widgets. By default, Image Widgets are created without the LV_OBJ_FLAG_CLICKABLE flag, but you can add it to make an Image Widget detect and emit LV_EVENT_CLICKED events if desired.

Further Reading

Learn more about *Base-Widget Events* emitted by all Widgets.

Learn more about *Eventi*.

6.14.5 Keys

No *Keys* are processed by Image Widgets.

Further Reading

Learn more about *Keys*.

6.14.6 Esempio

Immagine da variabile e simbolo

```
#include "../../lv_examples.h"
#if LV_USE_IMAGE && LV_BUILD_EXAMPLES

void lv_example_image_1(void)
{
    LV_IMAGE_DECLARE(img_cogwheel_argb);
    lv_obj_t * img1 = lv_image_create(lv_screen_active());
    lv_image_set_src(img1, &img_cogwheel_argb);
    lv_obj_align(img1, LV_ALIGN_CENTER, 0, 0);

    lv_obj_t * img2 = lv_image_create(lv_screen_active());
    lv_image_set_src(img2, LV_SYMBOL_OK "Accept");
    lv_obj_align_to(img2, img1, LV_ALIGN_OUT_BOTTOM_MID, 0, 20);
}

#endif
```

Ricolorazione dell'immagine

```
#include "../../lv_examples.h"
#if LV_USE_IMAGE && LV_USE_SLIDER && LV_BUILD_EXAMPLES

static lv_obj_t * create_slider(lv_color_t color);
static void slider_event_cb(lv_event_t * e);
```

(continues on next page)

(continua dalla pagina precedente)

```
static lv_obj_t * red_slider, * green_slider, * blue_slider, * intense_
↪slider;
static lv_obj_t * img1;

/**
 * Demonstrate runtime image re-coloring
 */
void lv_example_image_2(void)
{
    /*Create 4 sliders to adjust RGB color and re-color intensity*/
    red_slider = create_slider(lv_palette_main(LV_PALETTE_RED));
    green_slider = create_slider(lv_palette_main(LV_PALETTE_GREEN));
    blue_slider = create_slider(lv_palette_main(LV_PALETTE_BLUE));
    intense_slider = create_slider(lv_palette_main(LV_PALETTE_GREY));

    lv_slider_set_value(red_slider, LV_OPA_20, LV_ANIM_OFF);
    lv_slider_set_value(green_slider, LV_OPA_90, LV_ANIM_OFF);
    lv_slider_set_value(blue_slider, LV_OPA_60, LV_ANIM_OFF);
    lv_slider_set_value(intense_slider, LV_OPA_50, LV_ANIM_OFF);

    lv_obj_align(red_slider, LV_ALIGN_LEFT_MID, 25, 0);
    lv_obj_align_to(green_slider, red_slider, LV_ALIGN_OUT_RIGHT_MID, 25,␣
↪0);
    lv_obj_align_to(blue_slider, green_slider, LV_ALIGN_OUT_RIGHT_MID, 25,␣
↪0);
    lv_obj_align_to(intense_slider, blue_slider, LV_ALIGN_OUT_RIGHT_MID,␣
↪25, 0);

    /*Now create the actual image*/
    LV_IMAGE_DECLARE(img_cogwheel_argb);
    img1 = lv_image_create(lv_screen_active());
    lv_image_set_src(img1, &img_cogwheel_argb);
    lv_obj_align(img1, LV_ALIGN_RIGHT_MID, -20, 0);

    lv_obj_send_event(intense_slider, LV_EVENT_VALUE_CHANGED, NULL);
}

static void slider_event_cb(lv_event_t * e)
{
    LV_UNUSED(e);

    /*Recolor the image based on the sliders' values*/
    lv_color_t color  = lv_color_make((uint8_t)lv_slider_get_value(red_
↪slider),
                                      (uint8_t)lv_slider_get_value(green_
↪slider),
                                      (uint8_t)lv_slider_get_value(blue_
↪slider));
    lv_opa_t intense = (lv_opa_t)lv_slider_get_value(intense_slider);
    lv_obj_set_style_image_recolor_opa(img1, intense, 0);
    lv_obj_set_style_image_recolor(img1, color, 0);
}

static lv_obj_t * create_slider(lv_color_t color)
{
    lv_obj_t * slider = lv_slider_create(lv_screen_active());
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_slider_set_range(slider, 0, 255);
    lv_obj_set_size(slider, 10, 200);
    lv_obj_set_style_bg_color(slider, color, LV_PART_KNOB);
    lv_obj_set_style_bg_color(slider, lv_color_darken(color, LV_OPA_40),␣
↪LV_PART_INDICATOR);
    lv_obj_add_event_cb(slider, slider_event_cb, LV_EVENT_VALUE_CHANGED,␣
↪NULL);
    return slider;
}

#endif
```

Ruota e zoom

```
#include "../../lv_examples.h"
#if LV_USE_IMAGE && LV_BUILD_EXAMPLES

static void set_angle(void * img, int32_t v)
{
    lv_image_set_rotation((lv_obj_t *)img, v);
}

static void set_scale(void * img, int32_t v)
{
    lv_image_set_scale((lv_obj_t *)img, v);
}

/**
 * Show transformations (zoom and rotation) using a pivot point.
 */
void lv_example_image_3(void)
{
    LV_IMAGE_DECLARE(img_cogwheel_argb);

    /*Now create the actual image*/
    lv_obj_t * img = lv_image_create(lv_screen_active());
    lv_image_set_src(img, &img_cogwheel_argb);
    lv_obj_align(img, LV_ALIGN_CENTER, 50, 50);
    lv_image_set_pivot(img, 0, 0);    /*Rotate around the top left corner*/

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, img);
    lv_anim_set_exec_cb(&a, set_angle);
    lv_anim_set_values(&a, 0, 3600);
    lv_anim_set_duration(&a, 5000);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);
    lv_anim_start(&a);

    lv_anim_set_exec_cb(&a, set_scale);
    lv_anim_set_values(&a, 128, 256);
    lv_anim_set_reverse_duration(&a, 3000);
    lv_anim_start(&a);
}

#endif
```

Offset e stile dell'immagine

```
#include "../../lv_examples.h"
#if LV_USE_IMAGE && LV_BUILD_EXAMPLES

static void ofs_y_anim(void * img, int32_t v)
{
    lv_image_set_offset_y((lv_obj_t *)img, v);
}

/**
 * Image styling and offset
 */
void lv_example_image_4(void)
{
    LV_IMAGE_DECLARE(img_skew_strip);

    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_bg_color(&style, lv_palette_main(LV_PALETTE_YELLOW));
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_image_recolor_opa(&style, LV_OPA_COVER);
    lv_style_set_image_recolor(&style, lv_color_black());

    lv_obj_t * img = lv_image_create(lv_screen_active());
    lv_obj_add_style(img, &style, 0);
    lv_image_set_src(img, &img_skew_strip);
    lv_obj_set_size(img, 150, 100);
    lv_obj_center(img);

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, img);
    lv_anim_set_exec_cb(&a, ofs_y_anim);
    lv_anim_set_values(&a, 0, 100);
    lv_anim_set_duration(&a, 3000);
    lv_anim_set_reverse_duration(&a, 500);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);
    lv_anim_start(&a);

}

#endif
```

6.14.7 API

lv_api_map_v8.h

lv_api_map_v9_0.h

lv_api_map_v9_1.h

lv_draw_buf.h

lv_draw_image.h

lv_draw_image_private.h

lv_image.h

lv_image_cache.h

lv_image_decoder.h

lv_image_decoder_private.h

lv_image_dsc.h

lv_image_header_cache.h

lv_image_private.h

lv_imagebutton.h

lv_imagebutton_private.h

lv_obj_property_names.h

6.15 Image Button (lv_imagebutton)

6.15.1 Panoramica

The Image Button is very similar to the simple 'Button' Widget. The only difference is that it displays user-defined images for each state instead of drawing a rectangle. The list of states is covered below.

You can set a left, right and middle image, and the middle image will be repeated to match the width of the Widget.

6.15.2 Parts and Styles

- `LV_PART_MAIN` Refers to the image(s). If background style properties are used, a rectangle will be drawn behind the Image Button.

6.15.3 Utilizzo

Image sources

To set the image in a state, use the `lv_imagebutton_set_src`(`imagebutton, LV_IMAGEBUTTON_STATE_..., src_left, src_mid, src_right`).

The image sources work the same as described in the *Image Widget* except that "Symbols" are not supported by the Image Button. Any of the sources can be `NULL`. Typically the middle image should be one of the set image sources.

If only `src_mid` is specified, the width of the widget will be set automatically to the width of the image. However, if all three sources are set, the width needs to be set by the user (using e.g. `lv_obj_set_width`) and the middle image will be tiled to fill the given size.

The possible states are:

- `LV_IMAGEBUTTON_STATE_RELEASED`
- `LV_IMAGEBUTTON_STATE_PRESSED`
- `LV_IMAGEBUTTON_STATE_DISABLED`
- `LV_IMAGEBUTTON_STATE_CHECKED_RELEASED`
- `LV_IMAGEBUTTON_STATE_CHECKED_PRESSED`
- `LV_IMAGEBUTTON_STATE_CHECKED_DISABLED`

The image sources set for state `LV_IMAGEBUTTON_STATE_RELEASED` are used for any state that has not had image sources set for it. If an image sources have been set for other states, e.g. `LV_IMAGEBUTTON_STATE_PRESSED`, they will be used instead when the Image Button is in that state.

Setting State Programmatically

Instead of the regular *lv_obj_add_state()* and *lv_obj_remove_state()* functions, use *lv_imagebutton_set_state*(imagebutton, LV_IMAGEBUTTON_STATE_...) to set the state of Image Buttons.

6.15.4 Eventi

- *LV_EVENT_VALUE_CHANGED* Sent when Image Button's CHECKED state is toggled. This requires the Image Button's *LV_OBJ_FLAG_CHECKABLE* flag to be set.

> **Further Reading**
>
> Learn more about *Base-Widget Events* emitted by all Widgets.
>
> Learn more about *Eventi*.

6.15.5 Keys

- LV_KEY_RIGHT/UP Go to CHECKED state if *LV_OBJ_FLAG_CHECKABLE* is enabled.
- LV_KEY_LEFT/DOWN Go to un-CHECKED state if *LV_OBJ_FLAG_CHECKABLE* is enabled.
- *LV_KEY_ENTER* Clicks the Image Button

> **Further Reading**
>
> Learn more about *Keys*.

6.15.6 Esempio

Pulsante immagine semplice

```
#include "../../lv_examples.h"
#if LV_USE_IMAGEBUTTON && LV_BUILD_EXAMPLES

void lv_example_imagebutton_1(void)
{
    LV_IMAGE_DECLARE(imagebutton_left);
    LV_IMAGE_DECLARE(imagebutton_right);
    LV_IMAGE_DECLARE(imagebutton_mid);

    /*Create a transition animation on width transformation and recolor.*/
    static lv_style_prop_t tr_prop[] = {LV_STYLE_TRANSFORM_WIDTH, LV_STYLE_
↪IMAGE_RECOLOR_OPA, 0};
    static lv_style_transition_dsc_t tr;
    lv_style_transition_dsc_init(&tr, tr_prop, lv_anim_path_linear, 200, 0,
↪ NULL);

    static lv_style_t style_def;
    lv_style_init(&style_def);
    lv_style_set_text_color(&style_def, lv_color_white());
    lv_style_set_transition(&style_def, &tr);

    /*Darken the button when pressed and make it wider*/
    static lv_style_t style_pr;
    lv_style_init(&style_pr);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_set_image_recolor_opa(&style_pr, LV_OPA_30);
    lv_style_set_image_recolor(&style_pr, lv_color_black());
    lv_style_set_transform_width(&style_pr, 20);

    /*Create an image button*/
    lv_obj_t * imagebutton1 = lv_imagebutton_create(lv_screen_active());
    lv_imagebutton_set_src(imagebutton1, LV_IMAGEBUTTON_STATE_RELEASED, &
↪imagebutton_left, &imagebutton_mid,
                           &imagebutton_right);
    lv_obj_add_style(imagebutton1, &style_def, 0);
    lv_obj_add_style(imagebutton1, &style_pr, LV_STATE_PRESSED);

    lv_obj_set_width(imagebutton1, 100);
    lv_obj_align(imagebutton1, LV_ALIGN_CENTER, 0, 0);

    /*Create a label on the image button*/
    lv_obj_t * label = lv_label_create(imagebutton1);
    lv_label_set_text(label, "Button");
    lv_obj_align(label, LV_ALIGN_CENTER, 0, -4);
}

#endif
```

6.15.7 API

lv_imagebutton.h

lv_imagebutton_private.h

6.16 Keyboard (lv_keyboard)

6.16.1 Panoramica

The Keyboard Widget is a special *Button Matrix (lv_buttonmatrix)* with predefined keymaps and other features to provide an on-screen virtual keyboard to write text into a *Text Area (lv_textarea)*.

6.16.2 Parts and Styles

Similar to Button Matrix, the Keyboard Widget consists of 2 parts:

- `LV_PART_MAIN` The main part. Uses the *typical background style properties*
- `LV_PART_ITEMS` The buttons. Also uses the *typical background style properties* as well as *text* properties.

6.16.3 Utilizzo

Modes

Keyboards have the following modes:

- `LV_KEYBOARD_MODE_TEXT_LOWER` Display lower case letters
- `LV_KEYBOARD_MODE_TEXT_UPPER` Display upper case letters
- `LV_KEYBOARD_MODE_SPECIAL` Display special characters
- `LV_KEYBOARD_MODE_NUMBER` Display numbers, +/- sign, and decimal dot
- `LV_KEYBOARD_MODE_USER_1` through `LV_KEYBOARD_MODE_USER_4` User-defined modes.

The layouts of the TEXT modes contain "keys" to change mode.

To set the mode programmatically, use `lv_keyboard_set_mode(kb, mode)`. The default mode is `LV_KEYBOARD_MODE_TEXT_UPPER`.

Assign Text Area

You can assign a *Text area* to the Keyboard to automatically put the clicked characters there. To assign the Text Area, use `lv_keyboard_set_textarea(kb, text_area)`.

Key Pop-Overs

To enable key pop-overs on press, like on common Android and iOS keyboards, use `lv_keyboard_set_popovers(kb, true)`. Default control maps are preconfigured to only show the pop-overs on keys that produce a symbol (i.e. not on space). If you use a custom keymap (see below), set the `LV_BUTTONMATRIX_CTRL_POPOVER` flag for each key for which a pop-over should be shown.

Note that pop-overs for keys in the top row will draw outside the Widget boundaries. To account for this, reserve extra free space on top of the Keyboard or ensure that the Keyboard is added *after* any Widgets adjacent to its top boundary (placing it "above" those Widgets) so that pop-overs will be drawn over them.

Pop-overs currently are merely a visual effect and don't allow selecting additional characters such as accented characters yet.

New Keymap

You can specify a new map (layout) for the Keyboard with `lv_keyboard_set_map(kb, LV_KEYBOARD_MODE_..., kb_map, kb_ctrl)`. See Button Matrix's *Button map* section for more information about creating new maps.

Keep in mind that using following keywords in the map will have the same effect as with the original map:

- `LV_SYMBOL_OK` Send `LV_EVENT_READY` to the assigned Text Area.
- `LV_SYMBOL_CLOSE` or `LV_SYMBOL_KEYBOARD` Send `LV_EVENT_CANCEL` to the assigned Text Area.
- `LV_SYMBOL_BACKSPACE` Delete character to the left.
- `LV_SYMBOL_LEFT` Move cursor left.
- `LV_SYMBOL_RIGHT` Move cursor right.
- `LV_SYMBOL_NEW_LINE` New line.
- `"ABC"` Load upper-case map.
- `"abc"` Load lower-case map.
- `"1#"` Load number map.

Key customization

Each key can be customized by using draw events such as `LV_EVENT_DRAW_TASK_ADDED`.

Check out the examples to see how it can be used in the practice.

6.16.4 Eventi

- `LV_EVENT_VALUE_CHANGED` Sent when the button is pressed/released or repeated after long press. The event data contains the ID of the pressed/released button.
- `LV_EVENT_READY`: The *Ok* button was clicked.
- `LV_EVENT_CANCEL`: The *Close* button was clicked.

The Keyboard has a **default event handler** callback called `lv_keyboard_def_event_cb()`, which handles the button pressing, map changing, sending events to the assigned text area, etc. You can remove it and replace it with a custom event handler if you wish, or add an additional call-back of your own.

> **Nota**
>
> In LVGL v8.0 and newer, adding an event handler to the Keyboard does not remove the default event handler. This behavior differs from v7, where adding an event handler would replace the previous one.

> **Further Reading**
>
> Learn more about *Base-Widget Events* emitted by all Widgets.
>
> Learn more about *Eventi*.

6.16.5 Keys

- `LV_KEY_RIGHT/UP/LEFT/RIGHT` To navigate among the buttons, selecting the one navigated to.
- `LV_KEY_ENTER` To press/release the selected button.

> **Further Reading**
>
> Learn more about *Keys*.

6.16.6 Esempio

Tastiera con area di testo

```
#include "../../lv_examples.h"
#if LV_USE_KEYBOARD && LV_BUILD_EXAMPLES

static void ta_event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * ta = lv_event_get_target_obj(e);
    lv_obj_t * kb = (lv_obj_t *)lv_event_get_user_data(e);
    if(code == LV_EVENT_FOCUSED) {
        lv_keyboard_set_textarea(kb, ta);
        lv_obj_remove_flag(kb, LV_OBJ_FLAG_HIDDEN);
    }

    if(code == LV_EVENT_DEFOCUSED) {
        lv_keyboard_set_textarea(kb, NULL);
        lv_obj_add_flag(kb, LV_OBJ_FLAG_HIDDEN);
    }
}

void lv_example_keyboard_1(void)
{
    /*Create a keyboard to use it with an of the text areas*/
    lv_obj_t * kb = lv_keyboard_create(lv_screen_active());

    /*Create a text area. The keyboard will write here*/
    lv_obj_t * ta1;
```

(continues on next page)

(continua dalla pagina precedente)

```
    ta1 = lv_textarea_create(lv_screen_active());
    lv_obj_align(ta1, LV_ALIGN_TOP_LEFT, 10, 10);
    lv_obj_add_event_cb(ta1, ta_event_cb, LV_EVENT_ALL, kb);
    lv_textarea_set_placeholder_text(ta1, "Hello");
    lv_obj_set_size(ta1, 140, 80);

    lv_obj_t * ta2;
    ta2 = lv_textarea_create(lv_screen_active());
    lv_obj_align(ta2, LV_ALIGN_TOP_RIGHT, -10, 10);
    lv_obj_add_event_cb(ta2, ta_event_cb, LV_EVENT_ALL, kb);
    lv_obj_set_size(ta2, 140, 80);

    lv_keyboard_set_textarea(kb, ta1);

    /*The keyboard will show Arabic characters if they are enabled */
#if LV_USE_ARABIC_PERSIAN_CHARS && LV_FONT_DEJAVU_16_PERSIAN_HEBREW
    lv_obj_set_style_text_font(kb, &lv_font_dejavu_16_persian_hebrew, 0);
    lv_obj_set_style_text_font(ta1, &lv_font_dejavu_16_persian_hebrew, 0);
    lv_obj_set_style_text_font(ta2, &lv_font_dejavu_16_persian_hebrew, 0);
#endif
}
#endif
```

Tastiera con mappa personalizzata

```
#include "../../lv_examples.h"
#if LV_USE_KEYBOARD && LV_BUILD_EXAMPLES

void lv_example_keyboard_2(void)
{
    /*Create an AZERTY keyboard map*/
    static const char * kb_map[] = {"A", "Z", "E", "R", "T", "Y", "U", "I",
 ↪ "O", "P", LV_SYMBOL_BACKSPACE, "\n",
                                    "Q", "S", "D", "F", "G", "J", "K", "L",
 ↪ "M",  LV_SYMBOL_NEW_LINE, "\n",
                                    "W", "X", "C", "V", "B", "N", ",", ".",
 ↪ ":", "!", "?", "\n",
                                    LV_SYMBOL_CLOSE, " ",  " ", " ", LV_
 ↪SYMBOL_OK, NULL
                                   };

    /*Set the relative width of the buttons and other controls*/
    static const lv_buttonmatrix_ctrl_t kb_ctrl[] = {LV_BUTTONMATRIX_CTRL_
 ↪WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_
 ↪BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_
 ↪CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4,
 ↪ LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_
 ↪BUTTONMATRIX_CTRL_WIDTH_6,
                                                        LV_BUTTONMATRIX_CTRL_
 ↪WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_
 ↪BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_
 ↪CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4,
 ↪ LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_6,
                                                        LV_BUTTONMATRIX_CTRL_
 ↪WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_
 ↪BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_
```

(continues on next page)

(continua dalla pagina precedente)

```
↪CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4,
↪ LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_BUTTONMATRIX_CTRL_WIDTH_4, LV_
↪BUTTONMATRIX_CTRL_WIDTH_4,
                                                        LV_BUTTONMATRIX_CTRL_
↪WIDTH_2, (lv_buttonmatrix_ctrl_t)(LV_BUTTONMATRIX_CTRL_HIDDEN | LV_
↪BUTTONMATRIX_CTRL_WIDTH_2), LV_BUTTONMATRIX_CTRL_WIDTH_6, (lv_
↪buttonmatrix_ctrl_t)(LV_BUTTONMATRIX_CTRL_HIDDEN | LV_BUTTONMATRIX_CTRL_
↪WIDTH_2), LV_BUTTONMATRIX_CTRL_WIDTH_2
                                                        };

    /*Create a keyboard and add the new map as USER_1 mode*/
    lv_obj_t * kb = lv_keyboard_create(lv_screen_active());

    lv_keyboard_set_map(kb, LV_KEYBOARD_MODE_USER_1, kb_map, kb_ctrl);
    lv_keyboard_set_mode(kb, LV_KEYBOARD_MODE_USER_1);

    /*Create a text area. The keyboard will write here*/
    lv_obj_t * ta;
    ta = lv_textarea_create(lv_screen_active());
    lv_obj_align(ta, LV_ALIGN_TOP_MID, 0, 10);
    lv_obj_set_size(ta, lv_pct(90), 80);
    lv_obj_add_state(ta, LV_STATE_FOCUSED);

    lv_keyboard_set_textarea(kb, ta);
}
#endif
```

Tastiera con disegno

```
#include "../../lv_examples.h"
#if LV_USE_BUTTONMATRIX && LV_BUILD_EXAMPLES

static void event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);
    bool pressed = false;
    if(lv_keyboard_get_selected_button(obj) == base_dsc->id1 && lv_obj_has_
↪state(obj, LV_STATE_PRESSED)) {
        pressed = true;
    }

    /*When the keyboard draws the buttons...*/
    if(base_dsc->part == LV_PART_ITEMS) {
        /*Get a color based on the button's index*/
        lv_palette_t palette = (lv_palette_t)(base_dsc->id1 % LV_PALETTE_
↪LAST);
        lv_draw_fill_dsc_t * fill_draw_dsc = lv_draw_task_get_fill_
↪dsc(draw_task);
        if(fill_draw_dsc) {
            fill_draw_dsc->color = pressed ? lv_palette_darken(palette, 3)␣
↪: lv_palette_main(palette);
        }
        lv_draw_label_dsc_t * label_draw_dsc = lv_draw_task_get_label_
```

(continues on next page)

(continua dalla pagina precedente)

```
↪dsc(draw_task);
        if(label_draw_dsc) {
            /*For the OK symbol, draw a star instead*/
            if(lv_strcmp(label_draw_dsc->text, LV_SYMBOL_OK) == 0) {
                label_draw_dsc->opa = 0;    /*Hide the label*/

                LV_IMAGE_DECLARE(img_star);
                lv_image_header_t header;
                lv_result_t res = lv_image_decoder_get_info(&img_star, &
↪header);
                if(res != LV_RESULT_OK) return;

                lv_area_t a;
                lv_area_set(&a, 0, 0, header.w - 1, header.h - 1);
                lv_area_t draw_task_area;
                lv_draw_task_get_area(draw_task, &draw_task_area);
                lv_area_align(&draw_task_area, &a, LV_ALIGN_CENTER, 0, 0);

                lv_draw_image_dsc_t img_draw_dsc;
                lv_draw_image_dsc_init(&img_draw_dsc);
                img_draw_dsc.src = &img_star;
                lv_draw_image(base_dsc->layer, &img_draw_dsc, &a);
            }
            else {
                /*For the other labels just pick an lighter color*/
                label_draw_dsc->color = lv_palette_lighten(palette, 4);
            }
        }

    }
}

/**
 * Add custom drawer to the keyboard to customize the buttons one by one
 */
void lv_example_keyboard_3(void)
{
    lv_obj_t * kb = lv_keyboard_create(lv_screen_active());
    lv_obj_add_event_cb(kb, event_cb, LV_EVENT_DRAW_TASK_ADDED, NULL);
    lv_obj_add_flag(kb, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);
    lv_obj_center(kb);
}

#endif
```

6.16.7 API

lv_api_map_v8.h

lv_keyboard.h

lv_keyboard_private.h

lv_obj_property_names.h

6.17 Label (lv_label)

6.17.1 Panoramica

A Label is the Widget used to display text.

6.17.2 Parts and Styles

- *LV_PART_MAIN* Uses the *typical background* and text properties. Padding values can be used to add space between the text and the edges of the Label's background.
- *LV_PART_SCROLLBAR* The scrollbar that is shown when the text is larger than the Widget's size.
- *LV_PART_SELECTED* Tells the style of the *selected text*. Only `text_color` and `bg_color` style properties can be used.

6.17.3 Utilizzo

Set text

You can set the text on a Label at runtime with *lv_label_set_text*`(label, "New text")`. This will allocate a buffer dynamically, and the provided string will be copied into that buffer. Therefore, you don't need to keep the text you pass to *lv_label_set_text()* in scope after that function returns.

With *lv_label_set_text_fmt*`(label, fmt, ...)` printf formatting can be used to set the text. Example: *lv_label_set_text_fmt*`(label, "Value: %d", 15)`.

Labels are able to show text from a static character buffer as well. To do so, use *lv_label_set_text_static*`(label, "Text")`. In this case, the text is not stored in dynamic memory and the given buffer is used directly instead. This means that the contents of the character buffer *must* remain valid for the life of the label or until another buffer is set via one of the above functions.

`const` strings are safe to use with *lv_label_set_text_static()* since they are stored in ROM memory, which is always accessible.

> **Avvertimento**
>
> Do not use `const` strings with *lv_label_set_text_static()* when the Label is being used in `LV_LABEL_LONG_DOT` mode since the Label will attempt to do an in-place edit of the string. This will cause an MCU exception by attempting to modify program memory (ROM).

> **Attenzione**
>
> If your Label is updated with new strings rapidly (e.g. > 30X / second, such as RPM in a dashboard, or an ADC value), and the length of those strings changes frequently, it is advisable to:
>
> - allocate a static string buffer large enough contain the largest possible string,
> - update that buffer with the new strings only when they will make a visible difference for the end user, and
> - update the Label with *lv_label_set_text_static*`(label, buffer)` using that buffer.
>
> Reason: if you use *lv_label_set_text*`(label, new_text)`, a memory realloc() will be forced every time the length of the string changes. That MCU overhead can be avoided by doing the above.

Newline

Newline characters are handled automatically by the Label Widget. You can use `\n` to make a line break. For example: `"line1\nline2\n\nline4"`

Long modes

By default, the width and height of the Label are set to `LV_SIZE_CONTENT`. Thus, the size of the Label is automatically expanded to the text size + padding + border width. Otherwise, if the width or height are explicitly set (using e.g.`lv_obj_set_width()` or a layout), the lines wider than the Label's width can be manipulated according to several long mode policies. Similarly, the policies can be applied if the height of the text is greater than the height of the Label.

- `LV_LABEL_LONG_WRAP` Wrap lines that are too long. If the height is `LV_SIZE_CONTENT` the Label's height will be expanded, otherwise the text will be clipped. (Default)
- `LV_LABEL_LONG_DOT` Replaces the last 3 characters from bottom right corner of the Label with dots (`.`)
- `LV_LABEL_LONG_SCROLL` If the text is wider than the label, scroll it horizontally back and forth. If it's higher, scroll vertically. Only one direction is scrolled and horizontal scrolling has higher precedence.
- `LV_LABEL_LONG_SCROLL_CIRCULAR` If the text is wider than the Label, scroll it horizontally continuously. If it's higher, scroll vertically. Only one direction is scrolled and horizontal scrolling has higher precedence.
- `LV_LABEL_LONG_CLIP` Simply clip the parts of the text outside the Label.

You can specify the long mode with `lv_label_set_long_mode(label, LV_LABEL_LONG_...)`

Note that `LV_LABEL_LONG_DOT` manipulates the text buffer in-place in order to add/remove the dots. When `lv_label_set_text()` or `lv_label_set_array_text()` are used, a separate buffer is allocated and this implementation detail is unnoticed. This is not the case with `lv_label_set_text_static()`. The buffer you pass to `lv_label_set_text_static()` must be writable if you plan to use `LV_LABEL_LONG_DOT`.

Text recolor

In the text, you can use commands to recolor parts of the text. For example: `Write a #ff0000 red# word`. This feature can be enabled individually for each label by `lv_label_set_recolor(label, en)` function. In the context of word-wrapped text, any Recoloring started on a line will be terminated at the end of the line where the line is wrapped if it was not already terminated by an ending `#` in the text.

Text selection

If enabled by `LV_LABEL_TEXT_SELECTION` part of the text can be selected. It's similar to when you use your mouse on a PC to select text. The whole mechanism (click and select the text as you drag your finger/mouse) is implemented in *Text Area (lv_textarea)* and the Label Widget only allows programmatic text selection with `lv_label_get_text_selection_start(label, start_char_index)` and `lv_label_get_text_selection_end(label, end_char_index)`.

Text alignment

To horizontally align the lines of a Label the `text_align` style property can be used with `lv_obj_set_style_text_align()` or `lv_style_set_text_align()`, passing one of the `LV_TEXT_ALIGN_...` enumeration values. Note that this has a visible effect only if:

- the Label Widget's width is larger than the width of the longest line of text, and
- the text has multiple lines with different line lengths.

Very long text

LVGL can efficiently handle very long (e.g. > 40k characters) Labels by saving some extra data (~12 bytes) to speed up drawing. To enable this feature, set `LV_LABEL_LONG_TXT_HINT` to `1` in `lv_conf.h`.

Custom scrolling animations

Some aspects of the scrolling animations in long modes `LV_LABEL_LONG_SCROLL` and `LV_LABEL_LONG_SCROLL_CIRCULAR` can be customized by setting the Label's animation style property, using *`lv_style_set_anim()`*. It will be treated as a template which will be used to create the scroll animations.

Symbols

The Labels can display symbols alongside letters (or on their own). Read the *Font (lv_font)* section to learn more about symbols.

6.17.4 Eventi

No special events are sent by Label Widgets. By default, Label Widgets are created without the LV_OBJ_FLAG_CLICKABLE flag, but you can add it to make a Label Widget emit LV_EVENT_CLICKED events if desired.

> **Further Reading**
>
> Learn more about *Base-Widget Events* emitted by all Widgets.
>
> Learn more about *Eventi*.

6.17.5 Keys

No *Keys* are processed by Label Widgets.

> **Further Reading**
>
> Learn more about *Keys*.

6.17.6 Esempio

A capo automatico, ricolorazione e scorrimento

```
#include "../../lv_examples.h"
#if LV_USE_LABEL && LV_BUILD_EXAMPLES

/**
 * Show line wrap, re-color, line align and text scrolling.
 */
void lv_example_label_1(void)
{
    lv_obj_t * label1 = lv_label_create(lv_screen_active());
    lv_label_set_long_mode(label1, LV_LABEL_LONG_MODE_WRAP);     /*Break␣
↪the long lines*/
    lv_label_set_recolor(label1, true);                      /*Enable re-
↪coloring by commands in the text*/
    lv_label_set_text(label1, "#0000ff Re-color# #ff00ff words# #ff0000 of␣
↪a# label, align the lines to the center "
                      "and wrap long text automatically.");
    lv_obj_set_width(label1, 150);  /*Set smaller width to make the lines␣
↪wrap*/
    lv_obj_set_style_text_align(label1, LV_TEXT_ALIGN_CENTER, 0);
    lv_obj_align(label1, LV_ALIGN_CENTER, 0, -40);

    lv_obj_t * label2 = lv_label_create(lv_screen_active());
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_label_set_long_mode(label2, LV_LABEL_LONG_MODE_SCROLL_CIRCULAR);
↪ /*Circular scroll*/
    lv_obj_set_width(label2, 150);
    lv_label_set_text(label2, "It is a circularly scrolling text. ");
    lv_obj_align(label2, LV_ALIGN_CENTER, 0, 40);
}

#endif
```

Ombreggiatura del testo

```
#include "../../lv_examples.h"
#if LV_USE_LABEL && LV_BUILD_EXAMPLES

/**
 * Create a fake text shadow
 */
void lv_example_label_2(void)
{
    /*Create a style for the shadow*/
    static lv_style_t style_shadow;
    lv_style_init(&style_shadow);
    lv_style_set_text_opa(&style_shadow, LV_OPA_30);
    lv_style_set_text_color(&style_shadow, lv_color_black());

    /*Create a label for the shadow first (it's in the background)*/
    lv_obj_t * shadow_label = lv_label_create(lv_screen_active());
    lv_obj_add_style(shadow_label, &style_shadow, 0);

    /*Create the main label*/
    lv_obj_t * main_label = lv_label_create(lv_screen_active());
    lv_label_set_text(main_label, "A simple method to create\n"
                      "shadows on a text.\n"
                      "It even works with\n\n"
                      "newlines     and spaces.");

    /*Set the same text for the shadow label*/
    lv_label_set_text(shadow_label, lv_label_get_text(main_label));

    /*Position the main label*/
    lv_obj_align(main_label, LV_ALIGN_CENTER, 0, 0);

    /*Shift the second label down and to the right by 2 pixel*/
    lv_obj_align_to(shadow_label, main_label, LV_ALIGN_TOP_LEFT, 2, 2);
}

#endif
```

Visualizzazione di testi LTR, RTL e in cinese

```
#include "../../lv_examples.h"
#if LV_USE_LABEL && LV_BUILD_EXAMPLES && LV_FONT_DEJAVU_16_PERSIAN_HEBREW &
↪& LV_FONT_SOURCE_HAN_SANS_SC_16_CJK && LV_USE_BIDI

/**
```

(continues on next page)

(continua dalla pagina precedente)

```
 * Show mixed LTR, RTL and Chinese label
 */
void lv_example_label_3(void)
{
    lv_obj_t * ltr_label = lv_label_create(lv_screen_active());
    lv_label_set_text(ltr_label, "In modern terminology, a microcontroller␣
↪is similar to a system on a chip (SoC).");
    lv_obj_set_style_text_font(ltr_label, &lv_font_montserrat_16, 0);
    lv_obj_set_width(ltr_label, 310);
    lv_obj_align(ltr_label, LV_ALIGN_TOP_LEFT, 5, 5);

    lv_obj_t * rtl_label = lv_label_create(lv_screen_active());
    lv_label_set_text(rtl_label,
                      "□□□□, □□ □□□□ □□□□ □□□□□ □□□□□ □□□□□□ (□□□□□□□: CPU␣
↪- Central Processing Unit).");
    lv_obj_set_style_base_dir(rtl_label, LV_BASE_DIR_RTL, 0);
    lv_obj_set_style_text_font(rtl_label, &lv_font_dejavu_16_persian_
↪hebrew, 0);
    lv_obj_set_width(rtl_label, 310);
    lv_obj_align(rtl_label, LV_ALIGN_LEFT_MID, 5, 0);

    lv_obj_t * cz_label = lv_label_create(lv_screen_active());
    lv_label_set_text(cz_label,
                      "□□□□□□Embedded System□□\
↪n□□□□□□□
```

(continua dalla pagina precedente)

```
    lv_draw_label_dsc_init(&label_dsc);
    label_dsc.color = lv_color_white();
    label_dsc.align = LV_TEXT_ALIGN_CENTER;
    label_dsc.text = txt;
    label_dsc.font = &lv_font_montserrat_24;
    lv_area_t a = {0, 0, w - 1, h - 1};
    lv_draw_label(&layer, &label_dsc, &a);

    lv_canvas_finish_layer(canvas, &layer);

    lv_obj_delete(canvas);
}

/**
 * Draw label with gradient color
 */
void lv_example_label_4(void)
{
    /* Create the mask of a text by drawing it to a canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(mask, MASK_WIDTH, MASK_HEIGHT, LV_COLOR_
 ↪FORMAT_L8);
    LV_DRAW_BUF_INIT_STATIC(mask);

    generate_mask(&mask, MASK_WIDTH, MASK_HEIGHT, "Text with gradient");

    /* Create an object from where the text will be masked out.
     * Now it's a rectangle with a gradient but it could be an image too*/
    lv_obj_t * grad = lv_obj_create(lv_screen_active());
    lv_obj_set_size(grad, MASK_WIDTH, MASK_HEIGHT);
    lv_obj_center(grad);
    lv_obj_set_style_bg_color(grad, lv_color_hex(0xff0000), 0);
    lv_obj_set_style_bg_grad_color(grad, lv_color_hex(0x0000ff), 0);
    lv_obj_set_style_bg_grad_dir(grad, LV_GRAD_DIR_HOR, 0);
    lv_obj_set_style_bitmap_mask_src(grad, &mask, 0);
}

#endif
```

Personalizzazione dell'animazione a scorrimento circolare

```
#include "../../lv_examples.h"
#if LV_USE_LABEL && LV_BUILD_EXAMPLES

/**
 * Show customizing the circular scrolling animation of a label with `LV_
 ↪LABEL_LONG_MODE_SCROLL_CIRCULAR`
 * long mode.
 */
void lv_example_label_5(void)
{
    static lv_anim_t animation_template;
    static lv_style_t label_style;

    lv_anim_init(&animation_template);
    lv_anim_set_delay(&animation_template, 1000);           /*Wait 1␣
 ↪second to start the first scroll*/
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_anim_set_repeat_delay(&animation_template,
                             3000);    /*Repeat the scroll 3 seconds after␣
↪the label scrolls back to the initial position*/
    lv_anim_set_repeat_count(&animation_template, LV_ANIM_REPEAT_INFINITE);

    /*Initialize the label style with the animation template*/
    lv_style_init(&label_style);
    lv_style_set_anim(&label_style, &animation_template);

    lv_obj_t * label1 = lv_label_create(lv_screen_active());
    lv_label_set_long_mode(label1, LV_LABEL_LONG_MODE_SCROLL_CIRCULAR);    ␣
↪  /*Circular scroll*/
    lv_obj_set_width(label1, 150);
    lv_label_set_text(label1, "It is a circularly scrolling text. ");
    lv_obj_align(label1, LV_ALIGN_CENTER, 0, 40);
    lv_obj_add_style(label1, &label_style, LV_STATE_DEFAULT);           /
↪*Add the style to the label*/
}

#endif
```

Carattere a spaziatura fissa

```
#include "../../lv_examples.h"
#if LV_USE_LABEL && LV_BUILD_EXAMPLES && LV_FONT_MONTSERRAT_20

static bool fix_w_get_glyph_dsc(const lv_font_t * font, lv_font_glyph_dsc_
↪t * dsc, uint32_t letter,
                                uint32_t letter_next)
{
    bool ret = lv_font_get_glyph_dsc_fmt_txt(font, dsc, letter, letter_
↪next);
    if(!ret) return false;

    /* Set a fixed width */
    dsc->adv_w = 20;
    dsc->ofs_x = (dsc->adv_w - dsc->box_w) / 2;
    return true;
}

void lv_example_label_6(void)
{
    /* Clone the original font and override its behavior */
    static lv_font_t mono_font;
    mono_font = lv_font_montserrat_20;
    mono_font.get_glyph_dsc = fix_w_get_glyph_dsc;

    /* Create a label with normal font */
    lv_obj_t * label1 = lv_label_create(lv_screen_active());
    lv_obj_set_style_text_font(label1, &lv_font_montserrat_20, 0);
    lv_label_set_text(label1, "0123.Wabc");

    /* Create a label with fixed-width glyph descriptor override */
    lv_obj_t * label2 = lv_label_create(lv_screen_active());
    lv_obj_set_y(label2, 30);
    lv_obj_set_style_text_font(label2, &mono_font, 0);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_label_set_text(label2, "0123.Wabc");
}

#endif
```

6.17.7 API

lv_label.h

lv_label_private.h

lv_obj_property_names.h

lv_observer.h

6.18 LED (lv_led)

6.18.1 Panoramica

LEDs are rectangle-like (or circle) Widgets whose brightness can be adjusted. With lower brightness the color of the LED becomes darker.

6.18.2 Parts and Styles

- `LV_LED_PART_MAIN` uses the *typical background style properties*.

6.18.3 Utilizzo

Color

You can set the color of the LED with `lv_led_set_color(led, lv_color_hex(0xff0080))`. This will be used as its background color, border color, and shadow color.

Brightness

You can set their brightness with `lv_led_set_brightness(led, brightness)`. The `brightness` value should be in the range 0 (darkest) to 255 (lightest).

Toggle

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

You can set custom LED ON and OFF brightness values by defining macros `LV_LED_BRIGHT_MAX` and `LV_LED_BRIGHT_MIN` in your project. Their default values are 80 and 255. These too must be in the range [0..255].

6.18.4 Eventi

No special events are sent by LED Widgets.

Further Reading

Learn more about *Base-Widget Events* emitted by all Widgets.

Learn more about *Eventi*.

6.18.5 Keys

No *Keys* are processed by LED Widgets.

> **Further Reading**
>
> Learn more about *Keys*.

6.18.6 Esempio

LED con stile personalizzato

```
#include "../../lv_examples.h"
#if LV_USE_LED && LV_BUILD_EXAMPLES

/**
 * Create LED's with different brightness and color
 */
void lv_example_led_1(void)
{
    /*Create a LED and switch it OFF*/
    lv_obj_t * led1  = lv_led_create(lv_screen_active());
    lv_obj_align(led1, LV_ALIGN_CENTER, -80, 0);
    lv_led_off(led1);

    /*Copy the previous LED and set a brightness*/
    lv_obj_t * led2  = lv_led_create(lv_screen_active());
    lv_obj_align(led2, LV_ALIGN_CENTER, 0, 0);
    lv_led_set_brightness(led2, 150);
    lv_led_set_color(led2, lv_palette_main(LV_PALETTE_RED));

    /*Copy the previous LED and switch it ON*/
    lv_obj_t * led3  = lv_led_create(lv_screen_active());
    lv_obj_align(led3, LV_ALIGN_CENTER, 80, 0);
    lv_led_on(led3);
}

#endif
```

6.18.7 API

lv_led.h

lv_led_private.h

6.19 Line (lv_line)

6.19.1 Panoramica

The Line Widget is capable of drawing straight lines between a set of points.

6.19.2 Parts and Styles

- `LV_PART_MAIN` uses the *typical background* and line style properties.

6.19.3 Utilizzo

Set points

A Line's points have to be stored in an `lv_point_precise_t` array and passed to the Widget by the `lv_line_set_points(lines, point_array, point_cnt)` function.

Their coordinates can either be specified as raw pixel coordinates (e.g. `{5, 10}`), or as a percentage of the Line's bounding box using `lv_pct(x)`. In the latter case, the Line's width/height may need to be set explicitly using `lv_obj_set_width()` and `lv_obj_set_height()`, as percentage values do not automatically expand the bounding box.

Auto-size

By default, the Line's width and height are set to `LV_SIZE_CONTENT`. This means it will automatically set its size to fit all the points. If the size is set explicitly, parts on the Line may not be visible.

Invert y

By default, the *y == 0* point is at the top of the Widget. It might be counterintuitive in some cases, so the y coordinates can be inverted with `lv_line_set_y_invert(line, true)`. In this case, *y == 0* will be at the bottom of the Widget. *y invert* is disabled by default.

6.19.4 Eventi

Only *generic events* are sent by Line Widgets.

> **Further Reading**
>
> Learn more about *Base-Widget Events* emitted by all Widgets.
>
> Learn more about *Eventi*.

6.19.5 Keys

No *Keys* are processed by Line Widgets.

> **Further Reading**
>
> Learn more about *Keys*.

6.19.6 Esempio

Linea semplice

```
#include "../../lv_examples.h"
#if LV_USE_LINE && LV_BUILD_EXAMPLES

void lv_example_line_1(void)
{
    /*Create an array for the points of the line*/
    static lv_point_precise_t line_points[] = { {5, 5}, {70, 70}, {120, 10}
↪, {180, 60}, {240, 10} };

    /*Create style*/
    static lv_style_t style_line;
    lv_style_init(&style_line);
    lv_style_set_line_width(&style_line, 8);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_set_line_color(&style_line, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_line_rounded(&style_line, true);

    /*Create a line and apply the new style*/
    lv_obj_t * line1;
    line1 = lv_line_create(lv_screen_active());
    lv_line_set_points(line1, line_points, 5);     /*Set the points*/
    lv_obj_add_style(line1, &style_line, 0);
    lv_obj_center(line1);
}

#endif
```

6.19.7 API

lv_line.h

lv_line_private.h

6.20 List (lv_list)

6.20.1 Panoramica

The List Widget is basically a rectangle with vertical layout to which Buttons and Text can be added.

6.20.2 Parts and Styles

Background

- `LV_PART_MAIN` The main part of the List that uses the *typical background style properties*.
- `LV_PART_SCROLLBAR` The scrollbar. See *Base Widget (lv_obj)* documentation for details.

Buttons and Text

- See the *Button*'s and *Label*'s documentation.

6.20.3 Utilizzo

Buttons

`lv_list_add_button(list, icon, text)` adds a full-width button with an icon (that can be an image or symbol) and text. This function returns a pointer to the button created, which you can use to, for example, add an event call-back.

The text is scrolled horizontally if it is longer than the button.

Text

`lv_list_add_text(list, text)` adds a text string. This function returns a pointer to the label created, which you can use to, for example, change its text with one of the `lv_label_set_text...()` functions.

6.20.4 Eventi

No special events are sent by List Widgets, but events can be sent by Buttons as usual.

Further Reading

Learn more about *Base-Widget Events* emitted by all Widgets.

Learn more about *Eventi*.

6.20.5 Keys

No *Keys* are processed by List Widgets.

Further Reading

Learn more about *Keys*.

6.20.6 Esempio

Lista semplice

```
#include "../../lv_examples.h"
#if LV_USE_LIST && LV_BUILD_EXAMPLES
static lv_obj_t * list1;

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_CLICKED) {
        LV_UNUSED(obj);
        LV_LOG_USER("Clicked: %s", lv_list_get_button_text(list1, obj));
    }
}
void lv_example_list_1(void)
{
    /*Create a list*/
    list1 = lv_list_create(lv_screen_active());
    lv_obj_set_size(list1, 180, 220);
    lv_obj_center(list1);

    /*Add buttons to the list*/
    lv_obj_t * btn;
    lv_list_add_text(list1, "File");
    btn = lv_list_add_button(list1, LV_SYMBOL_FILE, "New");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);
    btn = lv_list_add_button(list1, LV_SYMBOL_DIRECTORY, "Open");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);
    btn = lv_list_add_button(list1, LV_SYMBOL_SAVE, "Save");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);
    btn = lv_list_add_button(list1, LV_SYMBOL_CLOSE, "Delete");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);
    btn = lv_list_add_button(list1, LV_SYMBOL_EDIT, "Edit");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);

    lv_list_add_text(list1, "Connectivity");
    btn = lv_list_add_button(list1, LV_SYMBOL_BLUETOOTH, "Bluetooth");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);
```

(continues on next page)

(continua dalla pagina precedente)

```
    btn = lv_list_add_button(list1, LV_SYMBOL_GPS, "Navigation");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);
    btn = lv_list_add_button(list1, LV_SYMBOL_USB, "USB");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);
    btn = lv_list_add_button(list1, LV_SYMBOL_BATTERY_FULL, "Battery");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);

    lv_list_add_text(list1, "Exit");
    btn = lv_list_add_button(list1, LV_SYMBOL_OK, "Apply");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);
    btn = lv_list_add_button(list1, LV_SYMBOL_CLOSE, "Close");
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);
}

#endif
```

Ordinamento di un elenco utilizzando i pulsanti Su e Giù

```
#include "../../lv_examples.h"
#if LV_USE_LIST && LV_BUILD_EXAMPLES

static lv_obj_t * list1;
static lv_obj_t * list2;

static lv_obj_t * currentButton = NULL;

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_CLICKED) {
        LV_LOG_USER("Clicked: %s", lv_list_get_button_text(list1, obj));

        if(currentButton == obj) {
            currentButton = NULL;
        }
        else {
            currentButton = obj;
        }
        lv_obj_t * parent = lv_obj_get_parent(obj);
        int32_t i;
        for(i = 0; i < (int32_t)lv_obj_get_child_count(parent); i++) {
            lv_obj_t * child = lv_obj_get_child(parent, i);
            if(child == currentButton) {
                lv_obj_add_state(child, LV_STATE_CHECKED);
            }
            else {
                lv_obj_remove_state(child, LV_STATE_CHECKED);
            }
        }
    }
}

static void event_handler_top(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
```

(continues on next page)

(continua dalla pagina precedente)

```
    if(code == LV_EVENT_CLICKED) {
        if(currentButton == NULL) return;
        lv_obj_move_background(currentButton);
        lv_obj_scroll_to_view(currentButton, LV_ANIM_ON);
    }
}

static void event_handler_up(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    if((code == LV_EVENT_CLICKED) || (code == LV_EVENT_LONG_PRESSED_
↪REPEAT)) {
        if(currentButton == NULL) return;
        int32_t index = lv_obj_get_index(currentButton);
        if(index <= 0) return;
        lv_obj_move_to_index(currentButton, index - 1);
        lv_obj_scroll_to_view(currentButton, LV_ANIM_ON);
    }
}

static void event_handler_center(lv_event_t * e)
{
    const lv_event_code_t code = lv_event_get_code(e);
    if((code == LV_EVENT_CLICKED) || (code == LV_EVENT_LONG_PRESSED_
↪REPEAT)) {
        if(currentButton == NULL) return;

        lv_obj_t * parent = lv_obj_get_parent(currentButton);
        const int32_t pos = (int32_t)lv_obj_get_child_count(parent) / 2;

        lv_obj_move_to_index(currentButton, pos);

        lv_obj_scroll_to_view(currentButton, LV_ANIM_ON);
    }
}

static void event_handler_dn(lv_event_t * e)
{
    const lv_event_code_t code = lv_event_get_code(e);
    if((code == LV_EVENT_CLICKED) || (code == LV_EVENT_LONG_PRESSED_
↪REPEAT)) {
        if(currentButton == NULL) return;
        const int32_t index = lv_obj_get_index(currentButton);

        lv_obj_move_to_index(currentButton, index + 1);
        lv_obj_scroll_to_view(currentButton, LV_ANIM_ON);
    }
}

static void event_handler_bottom(lv_event_t * e)
{
    const lv_event_code_t code = lv_event_get_code(e);
    if(code == LV_EVENT_CLICKED) {
        if(currentButton == NULL) return;
        lv_obj_move_foreground(currentButton);
        lv_obj_scroll_to_view(currentButton, LV_ANIM_ON);
```

(continues on next page)

(continua dalla pagina precedente)

```
    }
}

static void event_handler_swap(lv_event_t * e)
{
    const lv_event_code_t code = lv_event_get_code(e);
    if((code == LV_EVENT_CLICKED) || (code == LV_EVENT_LONG_PRESSED_
↪REPEAT)) {
        uint32_t cnt = lv_obj_get_child_count(list1);
        for(int i = 0; i < 100; i++) {
            lv_obj_t * obj = lv_obj_get_child(list1, (int32_t)lv_rand(0,␣
↪cnt - 1));
            lv_obj_move_to_index(obj, (int32_t)lv_rand(0, cnt - 1));
            if(currentButton != NULL) {
                lv_obj_scroll_to_view(currentButton, LV_ANIM_ON);
            }
        }
    }
}

void lv_example_list_2(void)
{
    /*Create a list*/
    list1 = lv_list_create(lv_screen_active());
    lv_obj_set_size(list1, lv_pct(60), lv_pct(100));
    lv_obj_set_style_pad_row(list1, 5, 0);

    /*Add buttons to the list*/
    lv_obj_t * btn;
    int i;
    for(i = 0; i < 15; i++) {
        btn = lv_button_create(list1);
        lv_obj_set_width(btn, lv_pct(50));
        lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);

        lv_obj_t * lab = lv_label_create(btn);
        lv_label_set_text_fmt(lab, "Item %d", i);
    }

    /*Select the first button by default*/
    currentButton = lv_obj_get_child(list1, 0);
    lv_obj_add_state(currentButton, LV_STATE_CHECKED);

    /*Create a second list with up and down buttons*/
    list2 = lv_list_create(lv_screen_active());
    lv_obj_set_size(list2, lv_pct(40), lv_pct(100));
    lv_obj_align(list2, LV_ALIGN_TOP_RIGHT, 0, 0);
    lv_obj_set_flex_flow(list2, LV_FLEX_FLOW_COLUMN);

    btn = lv_list_add_button(list2, NULL, "Top");
    lv_obj_add_event_cb(btn, event_handler_top, LV_EVENT_ALL, NULL);
    lv_group_remove_obj(btn);

    btn = lv_list_add_button(list2, LV_SYMBOL_UP, "Up");
    lv_obj_add_event_cb(btn, event_handler_up, LV_EVENT_ALL, NULL);
    lv_group_remove_obj(btn);
```

(continues on next page)

(continua dalla pagina precedente)

```
    btn = lv_list_add_button(list2, LV_SYMBOL_LEFT, "Center");
    lv_obj_add_event_cb(btn, event_handler_center, LV_EVENT_ALL, NULL);
    lv_group_remove_obj(btn);

    btn = lv_list_add_button(list2, LV_SYMBOL_DOWN, "Down");
    lv_obj_add_event_cb(btn, event_handler_dn, LV_EVENT_ALL, NULL);
    lv_group_remove_obj(btn);

    btn = lv_list_add_button(list2, NULL, "Bottom");
    lv_obj_add_event_cb(btn, event_handler_bottom, LV_EVENT_ALL, NULL);
    lv_group_remove_obj(btn);

    btn = lv_list_add_button(list2, LV_SYMBOL_SHUFFLE, "Shuffle");
    lv_obj_add_event_cb(btn, event_handler_swap, LV_EVENT_ALL, NULL);
    lv_group_remove_obj(btn);
}

#endif
```

6.20.7 API

lv_api_map_v8.h

lv_list.h

6.21 Lottie (lv_lottie)

6.21.1 Panoramica

The Lottie Widget is capable of parsing, rasterizing, and playing Lottie animations.

The Lottie animations are vector-based animations. Think of them as the modern combination of SVG and GIF.

The animations can be downloaded from various sources, such as https://lottiefiles.com/ or you can create your own animations using, for example, Adobe After Effects.

The Lottie Widget is based on *Canvas (lv_canvas)* because in order to render the animation the user needs to provide a buffer where the current animation frame is stored.

6.21.2 Parts and Styles

- *LV_PART_MAIN* The background of the Lottie animation. The typical background style properties apply but usually it is left transparent.

6.21.3 Utilizzo

Dipendenze

The Lottie Widget uses the ThorVG library which is integrated into LVGL. In order to use Lottie animations *LV_USE_THORVG_INTERNAL* (to use the built-in ThorVG) or *LV_USE_THORVG_EXTERNAL* (to link it externally) needs to be enabled in `lv_conf.h`. For vector graphics in general *LV_USE_VECTOR_GRAPHIC* also needs to be enabled.

As ThorVG is written in C++, when using *LV_USE_THORVG_INTERNAL* be sure that you can compile the cpp files.

Set a buffer

In order to render the animation a buffer needs to be assigned to the Lottie Widget. The animations are rendered in ARGB8888 format, therefore the buffer's size should be equal to `target_width x target_height x 4` bytes.

To keep the buffer size and the animation size consistent, the size of the Widget (i.e. the size of the animation) is set to the dimensions of the buffer internally.

The buffer can be set with either *`lv_lottie_set_buffer`*`(lottie, w, h, buf)` or *`lv_lottie_set_draw_buf`*`(lottie, draw_buf)`.

When a draw buffer is used, it must be already initialized by the user with *`LV_COLOR_FORMAT_ARGB8888_PREMULTIPLIED`* color format.

Set a source

`lv_example_lottie_approve.c` contains an example animation. Instead of storing the JSON string, a hex array is stored for the following reasons:

- to avoid escaping `"` character in the JSON file, and
- some compilers don't support very long strings.

`lvgl/scripts/filetohex.py` can be used to convert a Lottie file to a hex array. E.g.:

```
./filetohex.py path/to/lottie.json --filter-character --null-terminate >␣
↪out.txt
```

`--filter-character` filters out non-ASCII characters and `--null-terminate` makes sure that a trailing zero is appended to properly close the string.

To create an animation from data use *`lv_lottie_set_src_data`*`(lottie, data, sizeof(data))`

Lottie animations can be opened from JSON files by using *`lv_lottie_set_src_file`*`(lottie, "path/to/file.json")`. Note that the Lottie loader doesn't support LVGL's File System interface but a "normal path" should be used without a driver letter.

Get the animation

```
lv_anim_t * a = lv_lottie_get_anim(lottie)
```

returns the LVGL animation which controls the Lottie animation. By default it is running infinitely at 60FPS however the LVGL animation can be freely adjusted.

6.21.4 Eventi

No events are emitted by Lottie Widgets.

> **Further Reading**
>
> Learn more about *Base-Widget Events* emitted by all Widgets.
>
> Learn more about *Eventi*.

6.21.5 Keys

No keys are processed by Lottie Widgets.

> **Further Reading**
>
> Learn more about *Keys*.

6.21.6 Esempio

Caricamento di un'animazione Lottie da un array

```
#include "../../lv_examples.h"
#if LV_BUILD_EXAMPLES
#if LV_USE_LOTTIE

/**
 * Load an lottie animation from data
 */
void lv_example_lottie_1(void)
{
    extern const uint8_t lv_example_lottie_approve[];
    extern const size_t lv_example_lottie_approve_size;

    lv_obj_t * lottie = lv_lottie_create(lv_screen_active());
    lv_lottie_set_src_data(lottie, lv_example_lottie_approve, lv_example_
↪lottie_approve_size);

#if LV_DRAW_BUF_ALIGN == 4 && LV_DRAW_BUF_STRIDE_ALIGN == 1
    /*If there are no special requirements, just declare a buffer
      x4 because the Lottie is rendered in ARGB8888_PREMULTIPLIED format*/
    static uint8_t buf[64 * 64 * 4];
    lv_lottie_set_buffer(lottie, 64, 64, buf);
#else
    /*For GPUs and special alignment/strid setting use a draw_buf instead*/
    LV_DRAW_BUF_DEFINE(draw_buf, 64, 64, LV_COLOR_FORMAT_ARGB8888_
↪PREMULTIPLIED);
    lv_lottie_set_draw_buf(lottie, &draw_buf);
#endif

    lv_obj_center(lottie);
}

#else

void lv_example_lottie_1(void)
{
    /*fallback for online examples*/

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Lottie cannot be previewed online");
    lv_obj_center(label);
}

#endif /*LV_USE_LOTTIE*/

#endif /*LV_BUILD_EXAMPLES*/
```

Caricamento di un'animazione Lottie da un file

```
#include "../../lv_examples.h"
#if LV_BUILD_EXAMPLES
#if LV_USE_LOTTIE

/**
```

(continues on next page)

(continua dalla pagina precedente)

```
 * Load an lottie animation from file
 */
void lv_example_lottie_2(void)
{

    lv_obj_t * lottie = lv_lottie_create(lv_screen_active());
    lv_lottie_set_src_file(lottie, "lvgl/examples/widgets/lottie/lv_
↪example_lottie_approve.json");

#if LV_DRAW_BUF_ALIGN == 4 && LV_DRAW_BUF_STRIDE_ALIGN == 1
    /*If there are no special requirements, just declare a buffer
      x4 because the Lottie is rendered in ARGB8888_PREMULTIPLIED format*/
    static uint8_t buf[64 * 64 * 4];
    lv_lottie_set_buffer(lottie, 64, 64, buf);
#else
    /*For GPUs and special alignment/strid setting use a draw_buf instead*/
    LV_DRAW_BUF_DEFINE(draw_buf, 64, 64, LV_COLOR_FORMAT_ARGB8888_
↪PREMULTIPLIED);
    lv_lottie_set_draw_buf(lottie, &draw_buf);
#endif

    lv_obj_center(lottie);
}

#else

void lv_example_lottie_2(void)
{
    /*fallback for online examples*/

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Lottie cannot be previewed online");
    lv_obj_center(label);
}

#endif /*LV_USE_LOTTIE*/

#endif /*LV_BUILD_EXAMPLES*/
```

6.21.7 API

lv_lottie.h

lv_lottie_private.h

6.22 Menu (lv_menu)

6.22.1 Panoramica

The Menu Widget can be used to create multi-level menus that automatically handle navigation among menu levels, and enable its user to capture page navigation and click events.

6.22.2 Parts and Styles

The Menu Widget is built from the following Widgets:

- Main container: `lv_menu_main_cont`
- Main header: `lv_menu_main_header_cont`
- Back button: *Button (lv_button)*
- Back button icon: *Image (lv_image)*
- Main Page: `lv_menu_page`
- Sidebar container: `lv_menu_sidebar_cont`
- Sidebar header: `lv_menu_sidebar_header_cont`
- Back button: *Button (lv_button)*
- Back button icon: *Image (lv_image)*
- Sidebar Page: `lv_menu_page`

6.22.3 Utilizzo

Create a Menu

`lv_menu_create``(parent)` creates a new empty Menu.

This creates a Menu Widget with this object hierarchy:

- Menu: `lv_menu_t`
  - Hidden Sub-Page Storage: `lv_obj_t`
  - Main container: *`lv_menu_main_cont_class`*
    - Main header: *`lv_menu_main_header_cont_class`*
      - Back button: *Button (lv_button)*
      - Back button icon: *Image (lv_image)*
      - Main header title: *Label (lv_label)* (default hidden)

Note that no sidebar is created. You can create one later if you wish.

Header mode

The following header modes exist:

- *`LV_MENU_HEADER_TOP_FIXED`* Header is positioned at the top. (default)
- *`LV_MENU_HEADER_TOP_UNFIXED`* Header is positioned at the top and can be scrolled out of view.
- *`LV_MENU_HEADER_BOTTOM_FIXED`* Header is positioned at the bottom.

You can set header modes with *`lv_menu_set_mode_header`*`(menu, LV_MENU_HEADER...)`.

Root back button mode

The following root back button modes exist:

- `LV_MENU_ROOT_BACK_BTN_DISABLED` (default)
- `LV_MENU_ROOT_BACK_BTN_ENABLED`

You can set root back button modes with *`lv_menu_set_mode_root_back_button`*`(menu, LV_MENU_ROOT_BACK_BTN...)`.

Create a Menu Page

`lv_menu_page_create(menu, title)` creates a new empty Menu Page. You can add any Widgets to the Page.

Set a Menu Page in the main area

Once a Menu Page has been created, you can set it to the main area with `lv_menu_set_page(menu, page)`. `NULL` to clear main and clear Menu history.

Set a Menu Page in the sidebar

Once a Menu Page has been created, you can set it to the sidebar with `lv_menu_set_sidebar_page(menu, page)`. `NULL` to clear sidebar.

Linking between Menu Pages

If you have, for instance, created a button obj in the main Page. When you click the button Widget, you want it to open up a new Page, use `lv_menu_set_load_page_event(menu, btn, new page)`.

Create a Menu container, section, separator

The following objects can be created so that it is easier to style the Menu:

- `lv_menu_cont_create(parent_page)` creates a new empty container.
- `lv_menu_section_create(parent_page)` creates a new empty section.
- `lv_menu_separator_create(parent_page)` creates a separator.

6.22.4 Eventi

- `LV_EVENT_VALUE_CHANGED` Sent when a Page is shown.
  - `lv_menu_get_cur_main_page(menu)` returns a pointer to Menu Page that is currently displayed in the main container.
  - `lv_menu_get_cur_sidebar_page(menu)` returns a pointer to Menu Page that is currently displayed in the sidebar container.
- `LV_EVENT_CLICKED` Sent when a back button in a header from either main or sidebar is clicked. `LV_OBJ_FLAG_EVENT_BUBBLE` is enabled on the buttons so you can add events to the Menu itself.
  - `lv_menu_back_button_is_root(menu, button)` to check if button is root back button.

> **Further Reading**
>
> Learn more about *Base-Widget Events* emitted by all Widgets.
>
> Learn more about *Eventi*.

6.22.5 Keys

No *Keys* are processed by Menu Widgets.

> **Further Reading**
>
> Learn more about *Keys*.

6.22.6 Esempio

Menù semplice

```
#include "../../lv_examples.h"
#if LV_USE_MENU && LV_BUILD_EXAMPLES

void lv_example_menu_1(void)
{
    /*Create a menu object*/
    lv_obj_t * menu = lv_menu_create(lv_screen_active());
    lv_obj_set_size(menu, lv_display_get_horizontal_resolution(NULL), lv_
↪display_get_vertical_resolution(NULL));
    lv_obj_center(menu);

    lv_obj_t * cont;
    lv_obj_t * label;

    /*Create a sub page*/
    lv_obj_t * sub_page = lv_menu_page_create(menu, NULL);

    cont = lv_menu_cont_create(sub_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Hello, I am hiding here");

    /*Create a main page*/
    lv_obj_t * main_page = lv_menu_page_create(menu, NULL);

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 1");

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 2");

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 3 (Click me!)");
    lv_menu_set_load_page_event(menu, cont, sub_page);

    lv_menu_set_page(menu, main_page);
}

#endif
```

Menù semplice con pulsante principale

```
#include "../../lv_examples.h"
#if LV_USE_MENU && LV_USE_MSGBOX && LV_BUILD_EXAMPLES

static void back_event_handler(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    lv_obj_t * menu = (lv_obj_t *)lv_event_get_user_data(e);

    if(lv_menu_back_button_is_root(menu, obj)) {
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_obj_t * mbox1 = lv_msgbox_create(NULL);
        lv_msgbox_add_title(mbox1, "Hello");
        lv_msgbox_add_text(mbox1, "Root back btn click.");
        lv_msgbox_add_close_button(mbox1);
    }
}

void lv_example_menu_2(void)
{
    lv_obj_t * menu = lv_menu_create(lv_screen_active());
    lv_menu_set_mode_root_back_button(menu, LV_MENU_ROOT_BACK_BUTTON_
↪ENABLED);
    lv_obj_add_event_cb(menu, back_event_handler, LV_EVENT_CLICKED, menu);
    lv_obj_set_size(menu, lv_display_get_horizontal_resolution(NULL), lv_
↪display_get_vertical_resolution(NULL));
    lv_obj_center(menu);

    lv_obj_t * cont;
    lv_obj_t * label;

    /*Create a sub page*/
    lv_obj_t * sub_page = lv_menu_page_create(menu, NULL);

    cont = lv_menu_cont_create(sub_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Hello, I am hiding here");

    /*Create a main page*/
    lv_obj_t * main_page = lv_menu_page_create(menu, NULL);

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 1");

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 2");

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 3 (Click me!)");
    lv_menu_set_load_page_event(menu, cont, sub_page);

    lv_menu_set_page(menu, main_page);
}

#endif
```

Menù semplice con intestazione personalizzata

```
#include "../../lv_examples.h"
#if LV_USE_MENU && LV_BUILD_EXAMPLES

void lv_example_menu_3(void)
{
    /*Create a menu object*/
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_t * menu = lv_menu_create(lv_screen_active());
    lv_obj_set_size(menu, lv_display_get_horizontal_resolution(NULL), lv_
↪display_get_vertical_resolution(NULL));
    lv_obj_center(menu);

    /*Modify the header*/
    lv_obj_t * back_btn = lv_menu_get_main_header_back_button(menu);
    lv_obj_t * back_button_label = lv_label_create(back_btn);
    lv_label_set_text(back_button_label, "Back");

    lv_obj_t * cont;
    lv_obj_t * label;

    /*Create sub pages*/
    lv_obj_t * sub_1_page = lv_menu_page_create(menu, "Page 1");

    cont = lv_menu_cont_create(sub_1_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Hello, I am hiding here");

    lv_obj_t * sub_2_page = lv_menu_page_create(menu, "Page 2");

    cont = lv_menu_cont_create(sub_2_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Hello, I am hiding here");

    lv_obj_t * sub_3_page = lv_menu_page_create(menu, "Page 3");

    cont = lv_menu_cont_create(sub_3_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Hello, I am hiding here");

    /*Create a main page*/
    lv_obj_t * main_page = lv_menu_page_create(menu, NULL);

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 1 (Click me!)");
    lv_menu_set_load_page_event(menu, cont, sub_1_page);

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 2 (Click me!)");
    lv_menu_set_load_page_event(menu, cont, sub_2_page);

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 3 (Click me!)");
    lv_menu_set_load_page_event(menu, cont, sub_3_page);

    lv_menu_set_page(menu, main_page);
}

#endif
```

Menù semplice con pulsante mobile per aggiungere una nuova pagina di menu

```
#include "../../lv_examples.h"
#if LV_USE_MENU && LV_BUILD_EXAMPLES

static uint32_t btn_cnt = 1;
static lv_obj_t * main_page;
static lv_obj_t * menu;

static void float_button_event_cb(lv_event_t * e)
{
    LV_UNUSED(e);

    btn_cnt++;

    lv_obj_t * cont;
    lv_obj_t * label;

    lv_obj_t * sub_page = lv_menu_page_create(menu, NULL);

    cont = lv_menu_cont_create(sub_page);
    label = lv_label_create(cont);
    lv_label_set_text_fmt(label, "Hello, I am hiding inside %" LV_PRIu32"",
 ↪ btn_cnt);

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text_fmt(label, "Item %" LV_PRIu32"", btn_cnt);
    lv_menu_set_load_page_event(menu, cont, sub_page);

    lv_obj_scroll_to_view_recursive(cont, LV_ANIM_ON);
}

void lv_example_menu_4(void)
{
    /*Create a menu object*/
    menu = lv_menu_create(lv_screen_active());
    lv_obj_set_size(menu, lv_display_get_horizontal_resolution(NULL), lv_
 ↪display_get_vertical_resolution(NULL));
    lv_obj_center(menu);

    lv_obj_t * cont;
    lv_obj_t * label;

    /*Create a sub page*/
    lv_obj_t * sub_page = lv_menu_page_create(menu, NULL);

    cont = lv_menu_cont_create(sub_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Hello, I am hiding inside the first item");

    /*Create a main page*/
    main_page = lv_menu_page_create(menu, NULL);

    cont = lv_menu_cont_create(main_page);
    label = lv_label_create(cont);
    lv_label_set_text(label, "Item 1");
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_menu_set_load_page_event(menu, cont, sub_page);

    lv_menu_set_page(menu, main_page);

    /*Create floating btn*/
    lv_obj_t * float_btn = lv_button_create(lv_screen_active());
    lv_obj_set_size(float_btn, 50, 50);
    lv_obj_add_flag(float_btn, LV_OBJ_FLAG_FLOATING);
    lv_obj_align(float_btn, LV_ALIGN_BOTTOM_RIGHT, -10, -10);
    lv_obj_add_event_cb(float_btn, float_button_event_cb, LV_EVENT_CLICKED,
↪ menu);
    lv_obj_set_style_radius(float_btn, LV_RADIUS_CIRCLE, 0);
    lv_obj_set_style_bg_image_src(float_btn, LV_SYMBOL_PLUS, 0);
    lv_obj_set_style_text_font(float_btn, lv_theme_get_font_large(float_
↪btn), 0);
}

#endif
```

Menù complesso

```
#include "../../lv_examples.h"
#if LV_USE_MENU && LV_USE_MSGBOX && LV_BUILD_EXAMPLES

typedef enum {
    LV_MENU_ITEM_BUILDER_VARIANT_1,
    LV_MENU_ITEM_BUILDER_VARIANT_2
} lv_menu_builder_variant_t;

static void back_event_handler(lv_event_t * e);
static void switch_handler(lv_event_t * e);
static lv_obj_t * root_page;
static lv_obj_t * create_text(lv_obj_t * parent, const char * icon, const␣
↪char * txt,
                              lv_menu_builder_variant_t builder_variant);
static lv_obj_t * create_slider(lv_obj_t * parent,
                                const char * icon, const char * txt, int32_
↪t min, int32_t max, int32_t val);
static lv_obj_t * create_switch(lv_obj_t * parent,
                                const char * icon, const char * txt, bool␣
↪chk);

void lv_example_menu_5(void)
{
    lv_obj_t * menu = lv_menu_create(lv_screen_active());

    lv_color_t bg_color = lv_obj_get_style_bg_color(menu, LV_PART_MAIN);
    if(lv_color_brightness(bg_color) > 127) {
        lv_obj_set_style_bg_color(menu, lv_color_darken(lv_obj_get_style_
↪bg_color(menu, LV_PART_MAIN), 10), 0);
    }
    else {
        lv_obj_set_style_bg_color(menu, lv_color_darken(lv_obj_get_style_
↪bg_color(menu, LV_PART_MAIN), 50), 0);
    }
    lv_menu_set_mode_root_back_button(menu, LV_MENU_ROOT_BACK_BUTTON_
```

(continues on next page)

(continua dalla pagina precedente)

```
↪ENABLED);
    lv_obj_add_event_cb(menu, back_event_handler, LV_EVENT_CLICKED, menu);
    lv_obj_set_size(menu, lv_display_get_horizontal_resolution(NULL), lv_
↪display_get_vertical_resolution(NULL));
    lv_obj_center(menu);

    lv_obj_t * cont;
    lv_obj_t * section;

    /*Create sub pages*/
    lv_obj_t * sub_mechanics_page = lv_menu_page_create(menu, NULL);
    lv_obj_set_style_pad_hor(sub_mechanics_page, lv_obj_get_style_pad_
↪left(lv_menu_get_main_header(menu), LV_PART_MAIN), 0);
    lv_menu_separator_create(sub_mechanics_page);
    section = lv_menu_section_create(sub_mechanics_page);
    create_slider(section, LV_SYMBOL_SETTINGS, "Velocity", 0, 150, 120);
    create_slider(section, LV_SYMBOL_SETTINGS, "Acceleration", 0, 150, 50);
    create_slider(section, LV_SYMBOL_SETTINGS, "Weight limit", 0, 150, 80);

    lv_obj_t * sub_sound_page = lv_menu_page_create(menu, NULL);
    lv_obj_set_style_pad_hor(sub_sound_page, lv_obj_get_style_pad_left(lv_
↪menu_get_main_header(menu), LV_PART_MAIN), 0);
    lv_menu_separator_create(sub_sound_page);
    section = lv_menu_section_create(sub_sound_page);
    create_switch(section, LV_SYMBOL_AUDIO, "Sound", false);

    lv_obj_t * sub_display_page = lv_menu_page_create(menu, NULL);
    lv_obj_set_style_pad_hor(sub_display_page, lv_obj_get_style_pad_
↪left(lv_menu_get_main_header(menu), LV_PART_MAIN), 0);
    lv_menu_separator_create(sub_display_page);
    section = lv_menu_section_create(sub_display_page);
    create_slider(section, LV_SYMBOL_SETTINGS, "Brightness", 0, 150, 100);

    lv_obj_t * sub_software_info_page = lv_menu_page_create(menu, NULL);
    lv_obj_set_style_pad_hor(sub_software_info_page, lv_obj_get_style_pad_
↪left(lv_menu_get_main_header(menu), LV_PART_MAIN),
                             0);
    section = lv_menu_section_create(sub_software_info_page);
    create_text(section, NULL, "Version 1.0", LV_MENU_ITEM_BUILDER_VARIANT_
↪1);

    lv_obj_t * sub_legal_info_page = lv_menu_page_create(menu, NULL);
    lv_obj_set_style_pad_hor(sub_legal_info_page, lv_obj_get_style_pad_
↪left(lv_menu_get_main_header(menu), LV_PART_MAIN),
                             0);
    section = lv_menu_section_create(sub_legal_info_page);
    for(uint32_t i = 0; i < 15; i++) {
        create_text(section, NULL,
                    "This is a long long long long long long long long␣
↪long text, if it is long enough it may scroll.",
                    LV_MENU_ITEM_BUILDER_VARIANT_1);
    }

    lv_obj_t * sub_about_page = lv_menu_page_create(menu, NULL);
    lv_obj_set_style_pad_hor(sub_about_page, lv_obj_get_style_pad_left(lv_
↪menu_get_main_header(menu), LV_PART_MAIN), 0);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_menu_separator_create(sub_about_page);
    section = lv_menu_section_create(sub_about_page);
    cont = create_text(section, NULL, "Software information", LV_MENU_ITEM_
↪BUILDER_VARIANT_1);
    lv_menu_set_load_page_event(menu, cont, sub_software_info_page);
    cont = create_text(section, NULL, "Legal information", LV_MENU_ITEM_
↪BUILDER_VARIANT_1);
    lv_menu_set_load_page_event(menu, cont, sub_legal_info_page);

    lv_obj_t * sub_menu_mode_page = lv_menu_page_create(menu, NULL);
    lv_obj_set_style_pad_hor(sub_menu_mode_page, lv_obj_get_style_pad_
↪left(lv_menu_get_main_header(menu), LV_PART_MAIN), 0);
    lv_menu_separator_create(sub_menu_mode_page);
    section = lv_menu_section_create(sub_menu_mode_page);
    cont = create_switch(section, LV_SYMBOL_AUDIO, "Sidebar enable", true);
    lv_obj_add_event_cb(lv_obj_get_child(cont, 2), switch_handler, LV_
↪EVENT_VALUE_CHANGED, menu);

    /*Create a root page*/
    root_page = lv_menu_page_create(menu, "Settings");
    lv_obj_set_style_pad_hor(root_page, lv_obj_get_style_pad_left(lv_menu_
↪get_main_header(menu), LV_PART_MAIN), 0);
    section = lv_menu_section_create(root_page);
    cont = create_text(section, LV_SYMBOL_SETTINGS, "Mechanics", LV_MENU_
↪ITEM_BUILDER_VARIANT_1);
    lv_menu_set_load_page_event(menu, cont, sub_mechanics_page);
    cont = create_text(section, LV_SYMBOL_AUDIO, "Sound", LV_MENU_ITEM_
↪BUILDER_VARIANT_1);
    lv_menu_set_load_page_event(menu, cont, sub_sound_page);
    cont = create_text(section, LV_SYMBOL_SETTINGS, "Display", LV_MENU_
↪ITEM_BUILDER_VARIANT_1);
    lv_menu_set_load_page_event(menu, cont, sub_display_page);

    create_text(root_page, NULL, "Others", LV_MENU_ITEM_BUILDER_VARIANT_1);
    section = lv_menu_section_create(root_page);
    cont = create_text(section, NULL, "About", LV_MENU_ITEM_BUILDER_
↪VARIANT_1);
    lv_menu_set_load_page_event(menu, cont, sub_about_page);
    cont = create_text(section, LV_SYMBOL_SETTINGS, "Menu mode", LV_MENU_
↪ITEM_BUILDER_VARIANT_1);
    lv_menu_set_load_page_event(menu, cont, sub_menu_mode_page);

    lv_menu_set_sidebar_page(menu, root_page);

    lv_obj_send_event(lv_obj_get_child(lv_obj_get_child(lv_menu_get_cur_
↪sidebar_page(menu), 0), 0), LV_EVENT_CLICKED,
                      NULL);
}

static void back_event_handler(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    lv_obj_t * menu = (lv_obj_t *)lv_event_get_user_data(e);

    if(lv_menu_back_button_is_root(menu, obj)) {
        lv_obj_t * mbox1 = lv_msgbox_create(NULL);
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_msgbox_add_title(mbox1, "Hello");
        lv_msgbox_add_text(mbox1, "Root back btn click.");
        lv_msgbox_add_close_button(mbox1);
    }
}

static void switch_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * menu = (lv_obj_t *)lv_event_get_user_data(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        if(lv_obj_has_state(obj, LV_STATE_CHECKED)) {
            lv_menu_set_page(menu, NULL);
            lv_menu_set_sidebar_page(menu, root_page);
            lv_obj_send_event(lv_obj_get_child(lv_obj_get_child(lv_menu_
↪get_cur_sidebar_page(menu), 0), 0), LV_EVENT_CLICKED,
                              NULL);
        }
        else {
            lv_menu_set_sidebar_page(menu, NULL);
            lv_menu_clear_history(menu); /* Clear history because we will␣
↪be showing the root page later */
            lv_menu_set_page(menu, root_page);
        }
    }
}

static lv_obj_t * create_text(lv_obj_t * parent, const char * icon, const␣
↪char * txt,
                              lv_menu_builder_variant_t builder_variant)
{
    lv_obj_t * obj = lv_menu_cont_create(parent);

    lv_obj_t * img = NULL;
    lv_obj_t * label = NULL;

    if(icon) {
        img = lv_image_create(obj);
        lv_image_set_src(img, icon);
    }

    if(txt) {
        label = lv_label_create(obj);
        lv_label_set_text(label, txt);
        lv_label_set_long_mode(label, LV_LABEL_LONG_MODE_SCROLL_CIRCULAR);
        lv_obj_set_flex_grow(label, 1);
    }

    if(builder_variant == LV_MENU_ITEM_BUILDER_VARIANT_2 && icon && txt) {
        lv_obj_add_flag(img, LV_OBJ_FLAG_FLEX_IN_NEW_TRACK);
        lv_obj_swap(img, label);
    }

    return obj;
}
```

(continues on next page)

(continua dalla pagina precedente)

```
static lv_obj_t * create_slider(lv_obj_t * parent, const char * icon,␣
↪const char * txt, int32_t min, int32_t max,
                                int32_t val)
{
    lv_obj_t * obj = create_text(parent, icon, txt, LV_MENU_ITEM_BUILDER_
↪VARIANT_2);

    lv_obj_t * slider = lv_slider_create(obj);
    lv_obj_set_flex_grow(slider, 1);
    lv_slider_set_range(slider, min, max);
    lv_slider_set_value(slider, val, LV_ANIM_OFF);

    if(icon == NULL) {
        lv_obj_add_flag(slider, LV_OBJ_FLAG_FLEX_IN_NEW_TRACK);
    }

    return obj;
}

static lv_obj_t * create_switch(lv_obj_t * parent, const char * icon,␣
↪const char * txt, bool chk)
{
    lv_obj_t * obj = create_text(parent, icon, txt, LV_MENU_ITEM_BUILDER_
↪VARIANT_1);

    lv_obj_t * sw = lv_switch_create(obj);
    lv_obj_add_state(sw, chk ? LV_STATE_CHECKED : LV_STATE_DEFAULT);

    return obj;
}

#endif
```

6.22.7 API

lv_menu.h

lv_menu_private.h

6.23 Message Box (lv_msgbox)

6.23.1 Panoramica

Message boxes act as pop-ups. They are built from a content area with a helper to add text, an optional header (which can have a title, a close button, and other buttons), and an optional footer with buttons.

The text will be broken into multiple lines, and the height will be set automatically. If the height is set programmatically, the content will become scrollable.

The message box can be modal (blocking clicks on the rest of the screen) or not modal.

6.23.2 Parts and Styles

The message box is built from other Widgets, so you can check these Widgets' documentation for details.

- Content, header, and footer: *Base Widget (lv_obj)*

- Buttons: *Button (lv_button)*
- Title and content text: *Label (lv_label)*

6.23.3 Utilizzo

Create a message box

`lv_msgbox_create(parent)` creates a message box. If `parent` is `NULL` the message box will be modal, and will use the *Display di Default*'s Top *Layer* as a parent.

Adding buttons

If you want to add an [OK] or [Cancel] or other buttons for the user to have a choice of responses, add each button using the `lv_msgbox_add_footer_button(msgbox, btn_text)` function. Calling this function adds a footer (container) if one was not already present, and it returns a pointer to the button created, which can be used to add `LV_EVENT_CLICKED` (or other) events to detect and act on the user's response.

Footer buttons so added are evenly spaced and centered.

Buttons can also be added to the header if desired with `lv_msgbox_add_header_button(msgbox, symbol)`. Buttons so added are added to the right end of the header.

Getting the parts

The building blocks of the message box can be obtained using the following functions:

```
lv_obj_t * lv_msgbox_get_content(lv_obj_t * msgbox);
lv_obj_t * lv_msgbox_get_title(lv_obj_t * msgbox);
lv_obj_t * lv_msgbox_get_header(lv_obj_t * msgbox);
lv_obj_t * lv_msgbox_get_footer(lv_obj_t * msgbox);
```

Functions that add something to the message box return a pointer to the newly added Widget:

```
lv_obj_t * lv_msgbox_add_text(lv_obj_t * msgbox, const char * text);
lv_obj_t * lv_msgbox_add_title(lv_obj_t * msgbox, const char * title);
lv_obj_t * lv_msgbox_add_close_button(lv_obj_t * msgbox);
lv_obj_t * lv_msgbox_add_header_button(lv_obj_t * msgbox, const void *␣
↪symbol);
lv_obj_t * lv_msgbox_add_footer_button(lv_obj_t * msgbox, const char *␣
↪text);
```

Close the message box

`lv_msgbox_close(msgbox)` closes (deletes) the message box.

`lv_msgbox_close_async(msgbox)` closes (deletes) the message box asynchronously. This is useful if you want the message box to close the on the next call to `lv_timer_handler` instead of immediately.

6.23.4 Eventi

No special events are sent by Message Box Widgets. See these Widgets' documentation for details: *Button (lv_button)* and *Label (lv_label)*.

Further Reading

Learn more about *Base-Widget Events* emitted by all Widgets.

Learn more about *Eventi*.

6.23.5 Keys

No *Keys* are processed by Message Box Widgets.

> **Further Reading**
>
> Learn more about *Keys*.

6.23.6 Esempio

Casella messaggi semplice

```
#include "../../lv_examples.h"
#if LV_USE_MSGBOX && LV_BUILD_EXAMPLES

static void event_cb(lv_event_t * e)
{
    lv_obj_t * btn = lv_event_get_target_obj(e);
    lv_obj_t * label = lv_obj_get_child(btn, 0);
    LV_UNUSED(label);
    LV_LOG_USER("Button %s clicked", lv_label_get_text(label));
}

void lv_example_msgbox_1(void)
{
    lv_obj_t * mbox1 = lv_msgbox_create(NULL);

    lv_msgbox_add_title(mbox1, "Hello");

    lv_msgbox_add_text(mbox1, "This is a message box with two buttons.");
    lv_msgbox_add_close_button(mbox1);

    lv_obj_t * btn;
    btn = lv_msgbox_add_footer_button(mbox1, "Apply");
    lv_obj_add_event_cb(btn, event_cb, LV_EVENT_CLICKED, NULL);
    btn = lv_msgbox_add_footer_button(mbox1, "Cancel");
    lv_obj_add_event_cb(btn, event_cb, LV_EVENT_CLICKED, NULL);
    return;
}

#endif
```

Casella messaggi con scorrimento e stile

```
#include "../../lv_examples.h"
#if LV_USE_MSGBOX && LV_BUILD_EXAMPLES

static void minimize_button_event_cb(lv_event_t * e)
{
    lv_obj_t * mbox = (lv_obj_t *) lv_event_get_user_data(e);
    lv_obj_add_flag(mbox, LV_OBJ_FLAG_HIDDEN);
}

void lv_example_msgbox_2(void)
{
    lv_obj_t * setting = lv_msgbox_create(lv_screen_active());
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_style_clip_corner(setting, true, 0);

    /* setting fixed size */
    lv_obj_set_size(setting, 300, 200);

    /* setting's titlebar/header */
    lv_msgbox_add_title(setting, "Setting");
    lv_obj_t * minimize_button = lv_msgbox_add_header_button(setting, LV_
↪SYMBOL_MINUS);
    lv_obj_add_event_cb(minimize_button, minimize_button_event_cb, LV_
↪EVENT_CLICKED, setting);
    lv_msgbox_add_close_button(setting);

    /* setting's content*/
    lv_obj_t * content = lv_msgbox_get_content(setting);
    lv_obj_set_flex_flow(content, LV_FLEX_FLOW_COLUMN);
    lv_obj_set_flex_align(content, LV_FLEX_ALIGN_START, LV_FLEX_ALIGN_
↪CENTER, LV_FLEX_ALIGN_CENTER);
    lv_obj_set_style_pad_right(content, -1, LV_PART_SCROLLBAR);

    lv_obj_t * cont_brightness = lv_obj_create(content);
    lv_obj_set_size(cont_brightness, lv_pct(100), LV_SIZE_CONTENT);
    lv_obj_set_flex_flow(cont_brightness, LV_FLEX_FLOW_COLUMN);
    lv_obj_set_flex_align(cont_brightness, LV_FLEX_ALIGN_CENTER,  LV_FLEX_
↪ALIGN_START, LV_FLEX_ALIGN_CENTER);

    lv_obj_t * lb_brightness = lv_label_create(cont_brightness);
    lv_label_set_text(lb_brightness, "Brightness : ");
    lv_obj_t * slider_brightness = lv_slider_create(cont_brightness);
    lv_obj_set_width(slider_brightness, lv_pct(100));
    lv_slider_set_value(slider_brightness, 50, LV_ANIM_OFF);

    lv_obj_t * cont_speed = lv_obj_create(content);
    lv_obj_set_size(cont_speed, lv_pct(100), LV_SIZE_CONTENT);
    lv_obj_set_flex_flow(cont_speed, LV_FLEX_FLOW_COLUMN);
    lv_obj_set_flex_align(cont_speed, LV_FLEX_ALIGN_CENTER,  LV_FLEX_ALIGN_
↪START, LV_FLEX_ALIGN_CENTER);

    lv_obj_t * lb_speed = lv_label_create(cont_speed);
    lv_label_set_text(lb_speed, "Speed : ");
    lv_obj_t * slider_speed = lv_slider_create(cont_speed);
    lv_obj_set_width(slider_speed, lv_pct(100));
    lv_slider_set_value(slider_speed, 80, LV_ANIM_OFF);

    /* footer */
    lv_obj_t * apply_button = lv_msgbox_add_footer_button(setting, "Apply
↪");
    lv_obj_set_flex_grow(apply_button, 1);

    lv_obj_t * cancel_button = lv_msgbox_add_footer_button(setting, "Cancel
↪");
    lv_obj_set_flex_grow(cancel_button, 1);

    lv_obj_t * footer = lv_msgbox_get_footer(setting);
    lv_obj_set_style_bg_color(footer, lv_palette_main(LV_PALETTE_INDIGO),␣
↪0);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_style_bg_opa(footer, LV_OPA_100, 0);
}
#endif
```

6.23.7 API

lv_msgbox.h

lv_msgbox_private.h

6.24 Roller (lv_roller)

6.24.1 Panoramica

Roller allows the end user to select an item from a list by scrolling through it. The item in the middle is the selected item and normally stands out from the other items due to different styles applied to it.

6.24.2 Parts and Styles

- `LV_PART_MAIN` The background of the roller uses the *typical background* and text style properties.
  - Style `text_line_space` adjusts the space between list items. Use `lv_obj_set_style_text_line_space()` to set this value.
  - When the Roller is scrolled and doesn't stop exactly on an item, it will automatically scroll to the nearest valid item in `anim_time` milliseconds as specified in the `anim_duration` style. Use `lv_obj_set_style_anim_duration()` to set this value.
- `LV_PART_SELECTED` The selected item (displayed in the middle of the Roller). Besides the typical background properties, it uses text style properties to change the appearance of the text of the selected item.

6.24.3 Utilizzo

Setting the list items

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

`LV_ROLLER_MODE_INFINITE` makes the Roller circular.

You can select an option programmatically with `lv_roller_set_selected(roller, id, LV_ANIM_ON)`, where *id* is the zero-based index of the list item to be selected.

If you don't know the index of an option can also select an item with `lv_roller_set_selected_str(roller, str, LV_ANIM_ON)`, where *str* is the string equal to one of the list items.

Get selected option

To get the *index* of the currently selected item use `lv_roller_get_selected(roller)`.

`lv_roller_get_selected_str(roller, buf, buf_size)` will copy the name of the selected item to `buf`.

Visible rows

The number of visible rows can be adjusted with `lv_roller_set_visible_row_count(roller, num)`.

This function calculates the height with the current style. If the font, line space, border width, etc. of the Roller changes, this function needs to be called again.

6.24.4 Eventi

- *LV_EVENT_VALUE_CHANGED* Sent when a new list item is selected.

> **Further Reading**
>
> Learn more about *Base-Widget Events* emitted by all Widgets.
>
> Learn more about *Eventi*.

6.24.5 Keys

- LV_KEY_RIGHT/DOWN Select next option
- LV_KEY_LEFT/UP Select previous option
- LY_KEY_ENTER Accept the selected option (sends *LV_EVENT_VALUE_CHANGED* event)

> **Further Reading**
>
> Learn more about *Keys*.

6.24.6 Esempio

Roller semplice

```
#include "../../lv_examples.h"
#if LV_USE_ROLLER && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        char buf[32];
        lv_roller_get_selected_str(obj, buf, sizeof(buf));
        LV_LOG_USER("Selected month: %s\n", buf);
    }
}

/**
 * An infinite roller with the name of the months
 */
void lv_example_roller_1(void)
{
    lv_obj_t * roller1 = lv_roller_create(lv_screen_active());
    lv_roller_set_options(roller1,
                          "January\n"
                          "February\n"
                          "March\n"
                          "April\n"
                          "May\n"
                          "June\n"
                          "July\n"
                          "August\n"
                          "September\n"
                          "October\n"
```

(continues on next page)

(continua dalla pagina precedente)

```
                            "November\n"
                            "December",
                            LV_ROLLER_MODE_INFINITE);

    lv_roller_set_visible_row_count(roller1, 4);
    lv_obj_center(roller1);
    lv_obj_add_event_cb(roller1, event_handler, LV_EVENT_ALL, NULL);
}

#endif
```

Roller con stili

```
#include "../../lv_examples.h"
#if LV_USE_ROLLER && LV_FONT_MONTSERRAT_22 && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        char buf[32];
        lv_roller_get_selected_str(obj, buf, sizeof(buf));
        LV_LOG_USER("Selected value: %s", buf);
    }
}

/**
 * Roller with various alignments and larger text in the selected area
 */
void lv_example_roller_2(void)
{
    /*A style to make the selected option larger*/
    static lv_style_t style_sel;
    lv_style_init(&style_sel);
    lv_style_set_text_font(&style_sel, &lv_font_montserrat_22);
    lv_style_set_bg_color(&style_sel, lv_color_hex3(0xf88));
    lv_style_set_border_width(&style_sel, 2);
    lv_style_set_border_color(&style_sel, lv_color_hex3(0xf00));

    const char * opts = "1\n2\n3\n4\n5\n6\n7\n8\n9\n10";
    lv_obj_t * roller;

    /*A roller on the left with left aligned text, and custom width*/
    roller = lv_roller_create(lv_screen_active());
    lv_roller_set_options(roller, opts, LV_ROLLER_MODE_NORMAL);
    lv_roller_set_visible_row_count(roller, 2);
    lv_obj_set_width(roller, 100);
    lv_obj_add_style(roller, &style_sel, LV_PART_SELECTED);
    lv_obj_set_style_text_align(roller, LV_TEXT_ALIGN_LEFT, 0);
    lv_obj_set_style_bg_color(roller, lv_color_hex3(0x0f0), 0);
    lv_obj_set_style_bg_grad_color(roller, lv_color_hex3(0xafa), 0);
    lv_obj_set_style_bg_grad_dir(roller, LV_GRAD_DIR_VER, 0);
    lv_obj_align(roller, LV_ALIGN_LEFT_MID, 10, 0);
    lv_obj_add_event_cb(roller, event_handler, LV_EVENT_ALL, NULL);
    lv_roller_set_selected(roller, 2, LV_ANIM_OFF);
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*A roller on the middle with center aligned text, and auto (default)␣
↪width*/
    roller = lv_roller_create(lv_screen_active());
    lv_roller_set_options(roller, opts, LV_ROLLER_MODE_NORMAL);
    lv_roller_set_visible_row_count(roller, 3);
    lv_obj_add_style(roller, &style_sel, LV_PART_SELECTED);
    lv_obj_align(roller, LV_ALIGN_CENTER, 0, 0);
    lv_obj_add_event_cb(roller, event_handler, LV_EVENT_ALL, NULL);
    lv_roller_set_selected(roller, 5, LV_ANIM_OFF);

    /*A roller on the right with right aligned text, and custom width*/
    roller = lv_roller_create(lv_screen_active());
    lv_roller_set_options(roller, opts, LV_ROLLER_MODE_NORMAL);
    lv_roller_set_visible_row_count(roller, 4);
    lv_obj_set_width(roller, 80);
    lv_obj_add_style(roller, &style_sel, LV_PART_SELECTED);
    lv_obj_set_style_text_align(roller, LV_TEXT_ALIGN_RIGHT, 0);
    lv_obj_align(roller, LV_ALIGN_RIGHT_MID, -10, 0);
    lv_obj_add_event_cb(roller, event_handler, LV_EVENT_ALL, NULL);
    lv_roller_set_selected(roller, 8, LV_ANIM_OFF);
}

#endif
```

Aggiungere una maschera di dissolvenza al roller

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

#if LV_USE_ROLLER && LV_DRAW_SW_COMPLEX && LV_BUILD_EXAMPLES && 1

static void generate_mask(lv_draw_buf_t * mask)
{
    /*Create a "8 bit alpha" canvas and clear it*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, mask);
    lv_canvas_fill_bg(canvas, lv_color_white(), LV_OPA_TRANSP);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    /*Draw a label to the canvas. The result "image" will be used as mask*/
    lv_draw_rect_dsc_t rect_dsc;
    lv_draw_rect_dsc_init(&rect_dsc);
    rect_dsc.bg_grad.dir = LV_GRAD_DIR_VER;
    rect_dsc.bg_grad.stops[0].color = lv_color_black();
    rect_dsc.bg_grad.stops[1].color = lv_color_white();
    rect_dsc.bg_grad.stops[0].opa = LV_OPA_COVER;
    rect_dsc.bg_grad.stops[1].opa = LV_OPA_COVER;
    lv_area_t a = {0, 0, mask->header.w - 1, mask->header.h / 2 - 10};
    lv_draw_rect(&layer, &rect_dsc, &a);

    a.y1 = mask->header.h / 2 + 10;
    a.y2 = mask->header.h - 1;
    rect_dsc.bg_grad.stops[0].color = lv_color_white();
    rect_dsc.bg_grad.stops[1].color = lv_color_black();
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_draw_rect(&layer, &rect_dsc, &a);

    lv_canvas_finish_layer(canvas, &layer);

    /*Comment it to make the mask visible*/
    lv_obj_delete(canvas);
}

/**
 * Add a fade mask to roller.
 */
void lv_example_roller_3(void)
{
    lv_obj_set_style_bg_color(lv_screen_active(), lv_palette_main(LV_
↪PALETTE_BLUE_GREY), 0);

    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_bg_color(&style, lv_color_black());
    lv_style_set_text_color(&style, lv_color_white());
    lv_style_set_border_width(&style, 0);
    lv_style_set_radius(&style, 0);

    lv_obj_t * roller1 = lv_roller_create(lv_screen_active());
    lv_obj_add_style(roller1, &style, 0);
    lv_obj_set_style_bg_opa(roller1, LV_OPA_50, LV_PART_SELECTED);

    lv_roller_set_options(roller1,
                          "January\n"
                          "February\n"
                          "March\n"
                          "April\n"
                          "May\n"
                          "June\n"
                          "July\n"
                          "August\n"
                          "September\n"
                          "October\n"
                          "November\n"
                          "December",
                          LV_ROLLER_MODE_NORMAL);

    lv_obj_center(roller1);
    lv_roller_set_visible_row_count(roller1, 4);

    /* Create the mask to make the top and bottom part of roller faded.
     * The width and height are empirical values for simplicity*/
    LV_DRAW_BUF_DEFINE_STATIC(mask, 130, 150, LV_COLOR_FORMAT_L8);
    LV_DRAW_BUF_INIT_STATIC(mask);

    generate_mask(&mask);
    lv_obj_set_style_bitmap_mask_src(roller1, &mask, 0);
}
#endif
```

6.24.7 API

lv_api_map_v8.h

lv_obj_property_names.h

lv_observer.h

lv_roller.h

lv_roller_private.h

6.25 Scale (lv_scale)

6.25.1 Panoramica

Scale Widgets show linear or circular scales with configurable ranges, tick counts, placement, labeling, and subsections (*Sections*) with custom styling.

6.25.2 Parts and Styles

The Scale Widget has the following three parts:

- `LV_PART_MAIN` Main line --- the blue line in example image.
- `LV_PART_ITEMS` Minor ticks --- the red minor ticks in example image.
- `LV_PART_INDICATOR` Major ticks and their labels (if enabled) --- the green major ticks and pink labels in example image.

6.25.3 Utilizzo

Mode

When a Scale Widget is created, it starts out in MODE `LV_SCALE_MODE_HORIZONTAL_BOTTOM`. This makes the scale horizontal with tick marks below the line. If you need it to have a different shape, orientation or tick position, use `lv_scale_set_mode(scale, mode)`, where `mode` can be any of these values:

- `LV_SCALE_MODE_HORIZONTAL_TOP`
- `LV_SCALE_MODE_HORIZONTAL_BOTTOM`
- `LV_SCALE_MODE_VERTICAL_LEFT`
- `LV_SCALE_MODE_VERTICAL_RIGHT`
- `LV_SCALE_MODE_ROUND_INNER`
- `LV_SCALE_MODE_ROUND_OUTER`

Setting range

A Scale starts its life with a default numeric range of [0..100] and a default angular range of 270. You can change these ranges with:

- `lv_scale_set_range(scale, min, max)`, and

- `lv_scale_set_angle_range(scale, angle_range)`

where `min` and `max` will become the numeric low and high values for the Scale, and `angle_range` is the angle between the low and high ends of the Scale.

Tick drawing order

Normally ticks and their labels are drawn first and the main line is drawn next, giving the ticks and their labels the appearance of being underneath the main line when there is overlap. You can reverse this sequence if you wish, making the ticks and labels appear on top the main line, using `lv_scale_set_draw_ticks_on_top(scale, true)`. (This effect can be reversed by passing `false` instead.)

Example with with ticks and labels drawn *under* the main line (default):

Example with ticks and labels drawn *on top of* the main line:

Configuring ticks

You configure the major and minor ticks of a Scale by calling 2 functions:

- `lv_scale_set_total_tick_count(scale, total_tick_count)`, and
- `lv_scale_set_major_tick_every(scale, nth_tick)`.

If you want labels to be drawn with the major ticks, call `lv_scale_set_label_show(scale, true)`. (Pass `false` to hide them again.)

By default, the labels shown are the numeric values of the scale at the major tick points. Can you specify different label content by calling `lv_scale_set_text_src(scale, custom_labels)` where `custom_labels` is an array of string pointers. Esempio:

```
static char * custom_labels[3] = {"One", "Two", NULL};
```

Note that a `NULL` pointer is used to terminate the list.

The content of the buffers pointed to need to remain valid for the life of the Scale.

For a Scale in one of the `..._ROUND_...` modes, you can optionally get it to rotate the major-tick labels to match the rotation of the major ticks using `lv_obj_set_style_transform_rotation(scale, LV_SCALE_LABEL_ROTATE_MATCH_TICKS, LV_PART_INDICATOR)`.

Alternately, labels can be rotated by a fixed amount (for any Scale mode). This example rotates labels by 20 degrees: `lv_obj_set_style_transform_rotation(scale, 200, LV_PART_INDICATOR)`.

Or both of the above can be done at the same time: `lv_obj_set_style_transform_rotation(scale, LV_SCALE_LABEL_ROTATE_MATCH_TICKS + 200, LV_PART_INDICATOR)`.

Some labels of the Scale might be drawn upside down (to match the tick) if the Scale includes a certain angle range. If you don't want this, to automatically rotate the labels to keep them upright, an additional flag can be used. Labels that would be upside down are then rotated 180 `lv_obj_set_style_transform_rotation(scale, LV_SCALE_LABEL_ROTATE_MATCH_TICKS | LV_SCALE_LABEL_ROTATE_KEEP_UPRIGHT, LV_PART_INDICATOR)`. Labels can also be moved a fixed distance in X and Y pixels using `lv_obj_set_style_translate_x(scale, 10, LV_PART_INDICATOR)`.

> **Nota**
>
> The major tick value is calculated with the `lv_map` API (when not setting custom labels), this calculation takes into consideration the total number of ticks and the Scale range, so the label drawn can present rounding errors when the calculated value is a floating-point value.

The length of the ticks can be configured with the length Style property on the `LV_PART_INDICATOR` for major ticks and `LV_PART_ITEMS` for minor ticks. Example with local Style: `lv_obj_set_style_length(scale, 5, LV_PART_INDICATOR)` for major ticks and `lv_obj_set_style_length(scale, 5, LV_PART_ITEMS)` for minor ticks. The ticks can be padded in either direction (outwards or inwards) for `..._ROUND_...` Scales only with: `lv_obj_set_style_radial_offset(scale, 5, LV_PART_INDICATOR)` for major ticks and `lv_obj_set_style_radial_offset(scale, 5, LV_PART_ITEMS)` for minor. Using length and radial offset together allows total control of the tick position.

It is also possible to offset the labels from the major ticks (either positive or negative) using `lv_obj_set_style_pad_radial(scale, 5, LV_PART_INDICATOR)`

Sections

Sections make it possible for portions of a Scale to *convey meaning* by using different Style properties to draw them (colors, line thicknesses, font, etc.).

A Section represents a sub-range of the Scale, whose Styles (like Cascading Style Sheets) take precedence while drawing the PARTS (lines, arcs, ticks and labels) of the Scale that are within the range of that Section.

If a PART of a Scale is within the range of 2 or more Sections (i.e. those Sections overlap), the Style's properties belonging to the most recently added Section takes precedence over the same style properties of other Section(s) that "involve" that PART.

Creating Sections

A Section is created using `lv_scale_add_section(scale)`, which returns a pointer to a `lv_scale_section_t` object. This creates a Section with range [0..0] and no Styles added to it, which ensures that Section will not be drawn yet: it needs both a range inside the Scale's range and at least one *Style* added to it before it will be used in drawing the Scale.

Next, set the range using `lv_scale_section_set_range(section, min, max)` where `min` and `max` are the Section's boundary values that should normally be within the Scale's value range. (If they are only partially within the Scale's range, the Scale will only use that portion of the Section that overlaps the Scale's range. If a Section's range is not within the Scale's range at all, it will not be used in drawing. That can be useful to temporarily "disable" a Section, e.g. `lv_scale_section_set_range(section, 0, -1)`.)

Styling Sections

You set a Section's Style properties by creating a `lv_style_t` object for each "section" you want to appear different than the parent Scale. Add style properties as is documented in *Initialize Styles and Set/Get Properties*.

You attach each `lv_style_t` object to each Section it will apply to using `lv_scale_section_set_style(section, PART, style_pointer)`, where:

- `style_pointer` should point to the contents of a global or static variable (can be dynamically-allocated), since it needs to remain valid through the life of the Scale; and
- `PART` indicates which single *PART* of the parent Scale it will apply to, namely `LV_PART_MAIN`, `LV_PART_ITEMS` or `LV_PART_INDICATOR`.

Unlike adding normal styles to Widgets, you cannot combine PARTs by bit-wise OR-ing the PART values together to get the style to apply to more than one part. However, you can do something like this to accomplish the same thing:

```
static lv_style_t  tick_style;
lv_style_init(&tick_style);
lv_style_set_line_color(&tick_style, lv_palette_darken(LV_PALETTE_RED, 3));
lv_scale_section_set_style(section, LV_PART_ITEMS, &tick_style);
lv_scale_section_set_style(section, LV_PART_INDICATOR, &tick_style);
```

to get that one Style object to apply to both major and minor ticks.

`lv_style_t` objects can be shared among Sections and among PARTs, but unlike normal Styles added to a Widget, a Section can only have 1 style per PART. Thus, doing this:

```
lv_scale_section_set_style(section, LV_PART_INDICATOR, &tick_style_1);
lv_scale_section_set_style(section, LV_PART_INDICATOR, &tick_style_2);
```

replaces `tick_style_1` with `tick_style_2` for part `LV_PART_INDICATOR` rather than adding to it.

Useful Style Properties for Sections

The Style properties that are used during Scale drawing (and are thus useful) are:

- For main line *when it is a straight line* (`LV_PART_MAIN`):

  LV_STYLE_LINE_WIDTH
  `lv_style_set_line_width()`

  LV_STYLE_LINE_COLOR
  `lv_style_set_line_color()`

  LV_STYLE_LINE_OPA
  `lv_style_set_line_opa()`

- For main line *when it is an arc* (`LV_PART_MAIN`):

  LV_STYLE_ARC_WIDTH
  `lv_style_set_arc_width()`

  LV_STYLE_ARC_COLOR
  `lv_style_set_arc_color()`

  LV_STYLE_ARC_OPA
  `lv_style_set_arc_opa()`

  LV_STYLE_ARC_ROUNDED
  `lv_style_set_arc_rounded()`

  LV_STYLE_ARC_IMAGE_SRC
  `lv_style_set_arc_image_src()`

- For tick lines (`LV_PART_ITEMS` and `LV_PART_INDICATOR`):

  LV_STYLE_LINE_WIDTH
  `lv_style_set_line_width()`

  LV_STYLE_LINE_COLOR
  `lv_style_set_line_color()`

  LV_STYLE_LINE_OPA
  `lv_style_set_line_opa()`

- For labels on major ticks (`LV_PART_INDICATOR`)

  LV_STYLE_TEXT_COLOR
  `lv_style_set_text_color()`

  LV_STYLE_TEXT_OPA
  `lv_style_set_text_opa()`

  LV_STYLE_TEXT_LETTER_SPACE
  `lv_style_set_text_letter_space()`

  LV_STYLE_TEXT_FONT
  `lv_style_set_text_font()`

6.25.4 Eventi

No special events are sent by Scale Widgets.

In `LV_EVENT_DRAW_TASK_ADDED` events, a major or minor line draw descriptor's members `id1` and `id2` will be the tick index and tick value, respectively. If the part is `LV_PART_INDICATOR`, it is a major tick. If the part is `LV_PART_ITEMS` it is a minor tick.

Further Reading

Learn more about *Base-Widget Events* emitted by all Widgets.

Learn more about *Eventi*.

6.25.5 Keys

No *Keys* are processed by Scale Widgets.

Further Reading

Learn more about *Keys*.

6.25.6 Esempio

Una semplice scala orizzontale

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES

/**
 * A simple horizontal scale
 */
void lv_example_scale_1(void)
{
    lv_obj_t * scale = lv_scale_create(lv_screen_active());
    lv_obj_set_size(scale, lv_pct(80), 100);
    lv_scale_set_mode(scale, LV_SCALE_MODE_HORIZONTAL_BOTTOM);
    lv_obj_center(scale);

    lv_scale_set_label_show(scale, true);

    lv_scale_set_total_tick_count(scale, 31);
    lv_scale_set_major_tick_every(scale, 5);

    lv_obj_set_style_length(scale, 5, LV_PART_ITEMS);
    lv_obj_set_style_length(scale, 10, LV_PART_INDICATOR);
    lv_scale_set_range(scale, 10, 40);
}

#endif
```

Una scala verticale con sezioni e stile personalizzato

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES

/**
 * An vertical scale with section and custom styling
 */
void lv_example_scale_2(void)
{
    lv_obj_t * scale = lv_scale_create(lv_screen_active());
    lv_obj_set_size(scale, 60, 200);
    lv_scale_set_label_show(scale, true);
    lv_scale_set_mode(scale, LV_SCALE_MODE_VERTICAL_RIGHT);
    lv_obj_center(scale);

    lv_scale_set_total_tick_count(scale, 21);
    lv_scale_set_major_tick_every(scale, 5);

    lv_obj_set_style_length(scale, 10, LV_PART_INDICATOR);
    lv_obj_set_style_length(scale, 5, LV_PART_ITEMS);
    lv_scale_set_range(scale, 0, 100);

    static const char * custom_labels[] = {"0 °C", "25 °C", "50 °C", "75 °C
↪", "100 °C", NULL};
    lv_scale_set_text_src(scale, custom_labels);

    static lv_style_t indicator_style;
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_init(&indicator_style);

    /* Label style properties */
    lv_style_set_text_font(&indicator_style, LV_FONT_DEFAULT);
    lv_style_set_text_color(&indicator_style, lv_palette_darken(LV_PALETTE_
↪BLUE, 3));

    /* Major tick properties */
    lv_style_set_line_color(&indicator_style, lv_palette_darken(LV_PALETTE_
↪BLUE, 3));
    lv_style_set_width(&indicator_style, 10U);      /*Tick length*/
    lv_style_set_line_width(&indicator_style, 2U);  /*Tick width*/
    lv_obj_add_style(scale, &indicator_style, LV_PART_INDICATOR);

    static lv_style_t minor_ticks_style;
    lv_style_init(&minor_ticks_style);
    lv_style_set_line_color(&minor_ticks_style, lv_palette_lighten(LV_
↪PALETTE_BLUE, 2));
    lv_style_set_width(&minor_ticks_style, 5U);         /*Tick length*/
    lv_style_set_line_width(&minor_ticks_style, 2U);    /*Tick width*/
    lv_obj_add_style(scale, &minor_ticks_style, LV_PART_ITEMS);

    static lv_style_t main_line_style;
    lv_style_init(&main_line_style);
    /* Main line properties */
    lv_style_set_line_color(&main_line_style, lv_palette_darken(LV_PALETTE_
↪BLUE, 3));
    lv_style_set_line_width(&main_line_style, 2U); // Tick width
    lv_obj_add_style(scale, &main_line_style, LV_PART_MAIN);

    /* Add a section */
    static lv_style_t section_minor_tick_style;
    static lv_style_t section_label_style;
    static lv_style_t section_main_line_style;

    lv_style_init(&section_label_style);
    lv_style_init(&section_minor_tick_style);
    lv_style_init(&section_main_line_style);

    /* Label style properties */
    lv_style_set_text_font(&section_label_style, LV_FONT_DEFAULT);
    lv_style_set_text_color(&section_label_style, lv_palette_darken(LV_
↪PALETTE_RED, 3));

    lv_style_set_line_color(&section_label_style, lv_palette_darken(LV_
↪PALETTE_RED, 3));
    lv_style_set_line_width(&section_label_style, 5U); /*Tick width*/

    lv_style_set_line_color(&section_minor_tick_style, lv_palette_
↪lighten(LV_PALETTE_RED, 2));
    lv_style_set_line_width(&section_minor_tick_style, 4U); /*Tick width*/

    /* Main line properties */
    lv_style_set_line_color(&section_main_line_style, lv_palette_darken(LV_
↪PALETTE_RED, 3));
    lv_style_set_line_width(&section_main_line_style, 4U); /*Tick width*/
```

(continues on next page)

(continua dalla pagina precedente)

```
    /* Configure section styles */
    lv_scale_section_t * section = lv_scale_add_section(scale);
    lv_scale_set_section_range(scale, section, 75, 100);
    lv_scale_set_section_style_indicator(scale, section, &section_label_
↪style);
    lv_scale_set_section_style_items(scale, section, &section_minor_tick_
↪style);
    lv_scale_set_section_style_main(scale, section, &section_main_line_
↪style);


    lv_obj_set_style_bg_color(scale, lv_palette_main(LV_PALETTE_BLUE_GREY),
↪ 0);
    lv_obj_set_style_bg_opa(scale, LV_OPA_50, 0);
    lv_obj_set_style_pad_left(scale, 8, 0);
    lv_obj_set_style_radius(scale, 8, 0);
    lv_obj_set_style_pad_ver(scale, 20, 0);
}

#endif
```

Una semplice scala circolare

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES

LV_IMAGE_DECLARE(img_hand);

static lv_obj_t * needle_line;
static lv_obj_t * needle_img;

static void set_needle_line_value(void * obj, int32_t v)
{
    lv_scale_set_line_needle_value((lv_obj_t *)obj, needle_line, 60, v);
}

static void set_needle_img_value(void * obj, int32_t v)
{
    lv_scale_set_image_needle_value((lv_obj_t *)obj, needle_img, v);
}

/**
 * A simple round scale
 */
void lv_example_scale_3(void)
{
    lv_obj_t * scale_line = lv_scale_create(lv_screen_active());

    lv_obj_set_size(scale_line, 150, 150);
    lv_scale_set_mode(scale_line, LV_SCALE_MODE_ROUND_INNER);
    lv_obj_set_style_bg_opa(scale_line, LV_OPA_COVER, 0);
    lv_obj_set_style_bg_color(scale_line, lv_palette_lighten(LV_PALETTE_
↪RED, 5), 0);
    lv_obj_set_style_radius(scale_line, LV_RADIUS_CIRCLE, 0);
    lv_obj_set_style_clip_corner(scale_line, true, 0);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_align(scale_line, LV_ALIGN_LEFT_MID, LV_PCT(2), 0);

    lv_scale_set_label_show(scale_line, true);

    lv_scale_set_total_tick_count(scale_line, 31);
    lv_scale_set_major_tick_every(scale_line, 5);

    lv_obj_set_style_length(scale_line, 5, LV_PART_ITEMS);
    lv_obj_set_style_length(scale_line, 10, LV_PART_INDICATOR);
    lv_scale_set_range(scale_line, 10, 40);

    lv_scale_set_angle_range(scale_line, 270);
    lv_scale_set_rotation(scale_line, 135);

    needle_line = lv_line_create(scale_line);

    lv_obj_set_style_line_width(needle_line, 6, LV_PART_MAIN);
    lv_obj_set_style_line_rounded(needle_line, true, LV_PART_MAIN);

    lv_anim_t anim_scale_line;
    lv_anim_init(&anim_scale_line);
    lv_anim_set_var(&anim_scale_line, scale_line);
    lv_anim_set_exec_cb(&anim_scale_line, set_needle_line_value);
    lv_anim_set_duration(&anim_scale_line, 1000);
    lv_anim_set_repeat_count(&anim_scale_line, LV_ANIM_REPEAT_INFINITE);
    lv_anim_set_reverse_duration(&anim_scale_line, 1000);
    lv_anim_set_values(&anim_scale_line, 10, 40);
    lv_anim_start(&anim_scale_line);

    lv_obj_t * scale_img = lv_scale_create(lv_screen_active());

    lv_obj_set_size(scale_img, 150, 150);
    lv_scale_set_mode(scale_img, LV_SCALE_MODE_ROUND_INNER);
    lv_obj_set_style_bg_opa(scale_img, LV_OPA_COVER, 0);
    lv_obj_set_style_bg_color(scale_img, lv_palette_lighten(LV_PALETTE_RED,
↪ 5), 0);
    lv_obj_set_style_radius(scale_img, LV_RADIUS_CIRCLE, 0);
    lv_obj_set_style_clip_corner(scale_img, true, 0);
    lv_obj_align(scale_img, LV_ALIGN_RIGHT_MID, LV_PCT(-2), 0);

    lv_scale_set_label_show(scale_img, true);

    lv_scale_set_total_tick_count(scale_img, 31);
    lv_scale_set_major_tick_every(scale_img, 5);

    lv_obj_set_style_length(scale_img, 5, LV_PART_ITEMS);
    lv_obj_set_style_length(scale_img, 10, LV_PART_INDICATOR);
    lv_scale_set_range(scale_img, 10, 40);

    lv_scale_set_angle_range(scale_img, 270);
    lv_scale_set_rotation(scale_img, 135);

    /* image must point to the right. E.g. -O------>*/
    needle_img = lv_image_create(scale_img);
    lv_image_set_src(needle_img, &img_hand);
    lv_obj_align(needle_img, LV_ALIGN_CENTER, 47, -2);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_image_set_pivot(needle_img, 3, 4);

    lv_anim_t anim_scale_img;
    lv_anim_init(&anim_scale_img);
    lv_anim_set_var(&anim_scale_img, scale_img);
    lv_anim_set_exec_cb(&anim_scale_img, set_needle_img_value);
    lv_anim_set_duration(&anim_scale_img, 1000);
    lv_anim_set_repeat_count(&anim_scale_img, LV_ANIM_REPEAT_INFINITE);
    lv_anim_set_reverse_duration(&anim_scale_img, 1000);
    lv_anim_set_values(&anim_scale_img, 10, 40);
    lv_anim_start(&anim_scale_img);
}

#endif
```

Una scala circolare con sezione e stile personalizzato

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES

/**
 * A round scale with section and custom styling
 */
void lv_example_scale_4(void)
{
    lv_obj_t * scale = lv_scale_create(lv_screen_active());
    lv_obj_set_size(scale, 150, 150);
    lv_scale_set_label_show(scale, true);
    lv_scale_set_mode(scale, LV_SCALE_MODE_ROUND_OUTER);
    lv_obj_center(scale);

    lv_scale_set_total_tick_count(scale, 21);
    lv_scale_set_major_tick_every(scale, 5);

    lv_obj_set_style_length(scale, 5, LV_PART_ITEMS);
    lv_obj_set_style_length(scale, 10, LV_PART_INDICATOR);
    lv_scale_set_range(scale, 0, 100);

    static const char * custom_labels[] = {"0 °C", "25 °C", "50 °C", "75 °C
↪", "100 °C", NULL};
    lv_scale_set_text_src(scale, custom_labels);

    static lv_style_t indicator_style;
    lv_style_init(&indicator_style);

    /* Label style properties */
    lv_style_set_text_font(&indicator_style, LV_FONT_DEFAULT);
    lv_style_set_text_color(&indicator_style, lv_palette_darken(LV_PALETTE_
↪BLUE, 3));

    /* Major tick properties */
    lv_style_set_line_color(&indicator_style, lv_palette_darken(LV_PALETTE_
↪BLUE, 3));
    lv_style_set_width(&indicator_style, 10U);      /*Tick length*/
    lv_style_set_line_width(&indicator_style, 2U);  /*Tick width*/
    lv_obj_add_style(scale, &indicator_style, LV_PART_INDICATOR);
```

(continues on next page)

(continua dalla pagina precedente)

```
    static lv_style_t minor_ticks_style;
    lv_style_init(&minor_ticks_style);
    lv_style_set_line_color(&minor_ticks_style, lv_palette_lighten(LV_
↪PALETTE_BLUE, 2));
    lv_style_set_width(&minor_ticks_style, 5U);         /*Tick length*/
    lv_style_set_line_width(&minor_ticks_style, 2U);    /*Tick width*/
    lv_obj_add_style(scale, &minor_ticks_style, LV_PART_ITEMS);

    static lv_style_t main_line_style;
    lv_style_init(&main_line_style);
    /* Main line properties */
    lv_style_set_arc_color(&main_line_style, lv_palette_darken(LV_PALETTE_
↪BLUE, 3));
    lv_style_set_arc_width(&main_line_style, 2U); /*Tick width*/
    lv_obj_add_style(scale, &main_line_style, LV_PART_MAIN);

    /* Add a section */
    static lv_style_t section_minor_tick_style;
    static lv_style_t section_label_style;
    static lv_style_t section_main_line_style;

    lv_style_init(&section_label_style);
    lv_style_init(&section_minor_tick_style);
    lv_style_init(&section_main_line_style);

    /* Label style properties */
    lv_style_set_text_font(&section_label_style, LV_FONT_DEFAULT);
    lv_style_set_text_color(&section_label_style, lv_palette_darken(LV_
↪PALETTE_RED, 3));

    lv_style_set_line_color(&section_label_style, lv_palette_darken(LV_
↪PALETTE_RED, 3));
    lv_style_set_line_width(&section_label_style, 5U); /*Tick width*/

    lv_style_set_line_color(&section_minor_tick_style, lv_palette_
↪lighten(LV_PALETTE_RED, 2));
    lv_style_set_line_width(&section_minor_tick_style, 4U); /*Tick width*/

    /* Main line properties */
    lv_style_set_arc_color(&section_main_line_style, lv_palette_darken(LV_
↪PALETTE_RED, 3));
    lv_style_set_arc_width(&section_main_line_style, 4U); /*Tick width*/

    /* Configure section styles */
    lv_scale_section_t * section = lv_scale_add_section(scale);
    lv_scale_set_section_range(scale, section, 75, 100);
    lv_scale_set_section_style_indicator(scale, section, &section_label_
↪style);
    lv_scale_set_section_style_items(scale, section, &section_minor_tick_
↪style);
    lv_scale_set_section_style_main(scale, section, &section_main_line_
↪style);

}
```

(continues on next page)

(continua dalla pagina precedente)

```
#endif
```

Una scala con sezione e stile personalizzato

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES

/**
 * An scale with section and custom styling
 */
void lv_example_scale_5(void)
{
    lv_obj_t * scale = lv_scale_create(lv_screen_active());
    lv_obj_set_size(scale, lv_display_get_horizontal_resolution(NULL) / 2,␣
↪lv_display_get_vertical_resolution(NULL) / 2);
    lv_scale_set_label_show(scale, true);

    lv_scale_set_total_tick_count(scale, 10);
    lv_scale_set_major_tick_every(scale, 5);

    lv_obj_set_style_length(scale, 5, LV_PART_ITEMS);
    lv_obj_set_style_length(scale, 10, LV_PART_INDICATOR);
    lv_scale_set_range(scale, 25, 35);

    static const char * custom_labels[3] = {"One", "Two", NULL};
    lv_scale_set_text_src(scale, custom_labels);

    static lv_style_t indicator_style;
    lv_style_init(&indicator_style);
    /* Label style properties */
    lv_style_set_text_font(&indicator_style, LV_FONT_DEFAULT);
    lv_style_set_text_color(&indicator_style, lv_color_hex(0xff00ff));
    /* Major tick properties */
    lv_style_set_line_color(&indicator_style, lv_color_hex(0x00ff00));
    lv_style_set_width(&indicator_style, 10U); // Tick length
    lv_style_set_line_width(&indicator_style, 2U); // Tick width
    lv_obj_add_style(scale, &indicator_style, LV_PART_INDICATOR);

    static lv_style_t minor_ticks_style;
    lv_style_init(&minor_ticks_style);
    lv_style_set_line_color(&minor_ticks_style, lv_color_hex(0xff0000));
    lv_style_set_width(&minor_ticks_style, 5U); // Tick length
    lv_style_set_line_width(&minor_ticks_style, 2U); // Tick width
    lv_obj_add_style(scale, &minor_ticks_style, LV_PART_ITEMS);

    static lv_style_t main_line_style;
    lv_style_init(&main_line_style);
    /* Main line properties */
    lv_style_set_line_color(&main_line_style, lv_color_hex(0x0000ff));
    lv_style_set_line_width(&main_line_style, 2U); // Tick width
    lv_obj_add_style(scale, &main_line_style, LV_PART_MAIN);

    lv_obj_center(scale);

    /* Add a section */
    static lv_style_t section_minor_tick_style;
```

(continues on next page)

(continua dalla pagina precedente)

```
    static lv_style_t section_label_style;

    lv_style_init(&section_label_style);
    lv_style_init(&section_minor_tick_style);

    /* Label style properties */
    lv_style_set_text_font(&section_label_style, LV_FONT_DEFAULT);
    lv_style_set_text_color(&section_label_style, lv_color_hex(0xff0000));
    lv_style_set_text_letter_space(&section_label_style, 10);
    lv_style_set_text_opa(&section_label_style, LV_OPA_50);

    lv_style_set_line_color(&section_label_style, lv_color_hex(0xff0000));
    // lv_style_set_width(&section_label_style, 20U); // Tick length
    lv_style_set_line_width(&section_label_style, 5U); // Tick width

    lv_style_set_line_color(&section_minor_tick_style, lv_color_
↪hex(0x0000ff));
    // lv_style_set_width(&section_label_style, 20U); // Tick length
    lv_style_set_line_width(&section_minor_tick_style, 4U); // Tick width

    /* Configure section styles */
    lv_scale_section_t * section = lv_scale_add_section(scale);
    lv_scale_set_section_range(scale, section, 25, 30);
    lv_scale_set_section_style_indicator(scale, section, &section_label_
↪style);
    lv_scale_set_section_style_items(scale, section, &section_minor_tick_
↪style);
}

#endif
```

Una scala circolare con più lancette, simile a un orologio

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES

#if LV_USE_FLOAT
    #define my_PRIprecise "f"
#else
    #define my_PRIprecise LV_PRId32
#endif

static lv_obj_t * scale;
static lv_obj_t * minute_hand;
static lv_obj_t * hour_hand;
static lv_point_precise_t minute_hand_points[2];
static int32_t hour;
static int32_t minute;

static void timer_cb(lv_timer_t * timer)
{
    LV_UNUSED(timer);

    minute++;
    if(minute > 59) {
        minute = 0;
```

(continues on next page)

(continua dalla pagina precedente)

```
        hour++;
        if(hour > 11) {
            hour = 0;
        }
    }

    /**
     * the scale will store the needle line points in the existing
     * point array if one was set with `lv_line_set_points_mutable`.
     * Otherwise, it will allocate the needle line points.
     */

    /* the scale will store the minute hand line points in `minute_hand_
↪points` */
    lv_scale_set_line_needle_value(scale, minute_hand, 60, minute);
    /* log the points that were stored in the array */
    LV_LOG_USER(
        "minute hand points - "
        "0: (%" my_PRIprecise ", %" my_PRIprecise "), "
        "1: (%" my_PRIprecise ", %" my_PRIprecise ")",
        minute_hand_points[0].x, minute_hand_points[0].y,
        minute_hand_points[1].x, minute_hand_points[1].y
    );

    /* the scale will allocate the hour hand line points */
    lv_scale_set_line_needle_value(scale, hour_hand, 40, hour * 5 +␣
↪(minute / 12));
}

/**
 * A round scale with multiple needles, resembling a clock
 */
void lv_example_scale_6(void)
{
    scale = lv_scale_create(lv_screen_active());

    lv_obj_set_size(scale, 150, 150);
    lv_scale_set_mode(scale, LV_SCALE_MODE_ROUND_INNER);
    lv_obj_set_style_bg_opa(scale, LV_OPA_60, 0);
    lv_obj_set_style_bg_color(scale, lv_color_black(), 0);
    lv_obj_set_style_radius(scale, LV_RADIUS_CIRCLE, 0);
    lv_obj_set_style_clip_corner(scale, true, 0);
    lv_obj_center(scale);

    lv_scale_set_label_show(scale, true);

    lv_scale_set_total_tick_count(scale, 61);
    lv_scale_set_major_tick_every(scale, 5);

    static const char * hour_ticks[] = {"12", "1", "2", "3", "4", "5", "6",
↪ "7", "8", "9", "10", "11", NULL};
    lv_scale_set_text_src(scale, hour_ticks);

    static lv_style_t indicator_style;
    lv_style_init(&indicator_style);
```

(continues on next page)

(continua dalla pagina precedente)

```
    /* Label style properties */
    lv_style_set_text_font(&indicator_style, LV_FONT_DEFAULT);
    lv_style_set_text_color(&indicator_style, lv_palette_main(LV_PALETTE_
↪YELLOW));

    /* Major tick properties */
    lv_style_set_line_color(&indicator_style, lv_palette_main(LV_PALETTE_
↪YELLOW));
    lv_style_set_length(&indicator_style, 8); /* tick length */
    lv_style_set_line_width(&indicator_style, 2); /* tick width */
    lv_obj_add_style(scale, &indicator_style, LV_PART_INDICATOR);

    /* Minor tick properties */
    static lv_style_t minor_ticks_style;
    lv_style_init(&minor_ticks_style);
    lv_style_set_line_color(&minor_ticks_style, lv_palette_main(LV_PALETTE_
↪YELLOW));
    lv_style_set_length(&minor_ticks_style, 6); /* tick length */
    lv_style_set_line_width(&minor_ticks_style, 2); /* tick width */
    lv_obj_add_style(scale, &minor_ticks_style, LV_PART_ITEMS);

    /* Main line properties */
    static lv_style_t main_line_style;
    lv_style_init(&main_line_style);
    lv_style_set_arc_color(&main_line_style, lv_color_black());
    lv_style_set_arc_width(&main_line_style, 5);
    lv_obj_add_style(scale, &main_line_style, LV_PART_MAIN);

    lv_scale_set_range(scale, 0, 60);

    lv_scale_set_angle_range(scale, 360);
    lv_scale_set_rotation(scale, 270);

    minute_hand = lv_line_create(scale);
    lv_line_set_points_mutable(minute_hand, minute_hand_points, 2);

    lv_obj_set_style_line_width(minute_hand, 3, 0);
    lv_obj_set_style_line_rounded(minute_hand, true, 0);
    lv_obj_set_style_line_color(minute_hand, lv_color_white(), 0);

    hour_hand = lv_line_create(scale);

    lv_obj_set_style_line_width(hour_hand, 5, 0);
    lv_obj_set_style_line_rounded(hour_hand, true, 0);
    lv_obj_set_style_line_color(hour_hand, lv_palette_main(LV_PALETTE_RED),
↪ 0);

    hour = 11;
    minute = 5;
    lv_timer_t * timer = lv_timer_create(timer_cb, 250, NULL);
    lv_timer_ready(timer);
}

#endif
```

Personalizzazione del colore dell'etichetta principale della scala con l'evento LV_EVENT_DRAW_TASK_ADDED

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES

#include "../../../lvgl_private.h" /*To expose the fields of lv_draw_task_
↪t*/

static void draw_event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);
    lv_draw_label_dsc_t * label_draw_dsc = lv_draw_task_get_label_dsc(draw_
↪task);
    if(base_dsc->part == LV_PART_INDICATOR) {
        if(label_draw_dsc) {
            const lv_color_t color_idx[7] = {
                lv_palette_main(LV_PALETTE_RED),
                lv_palette_main(LV_PALETTE_ORANGE),
                lv_palette_main(LV_PALETTE_YELLOW),
                lv_palette_main(LV_PALETTE_GREEN),
                lv_palette_main(LV_PALETTE_CYAN),
                lv_palette_main(LV_PALETTE_BLUE),
                lv_palette_main(LV_PALETTE_PURPLE),
            };
            uint32_t major_tick = lv_scale_get_major_tick_every(obj);
            label_draw_dsc->color = color_idx[base_dsc->id1 / major_tick];

            /*Free the previously allocated text if needed*/
            if(label_draw_dsc->text_local) lv_free((void *)label_draw_dsc->
↪text);

            /*Malloc the text and set text_local as 1 to make LVGL␣
↪automatically free the text.
             * (Local texts are malloc'd internally by LVGL. Mimic this␣
↪behavior here too)*/
            char tmp_buffer[20] = {0}; /* Big enough buffer */
            lv_snprintf(tmp_buffer, sizeof(tmp_buffer), "%.1f",␣
↪(double)base_dsc->id2);
            label_draw_dsc->text = lv_strdup(tmp_buffer);
            label_draw_dsc->text_local = 1;

            lv_point_t size;
            lv_text_get_size(&size, label_draw_dsc->text, label_draw_dsc->
↪font, 0, 0, 1000, LV_TEXT_FLAG_NONE);
            int32_t new_w = size.x;
            int32_t old_w = lv_area_get_width(&draw_task->area);

            /* Distribute the new size equally on both sides */
            draw_task->area.x1 -= (new_w - old_w) / 2;
            draw_task->area.x2 += ((new_w - old_w) + 1) / 2;  /* +1 for␣
↪rounding */

        }
```

(continues on next page)

(continua dalla pagina precedente)

```
    }
}

/**
 * Customizing scale major tick label color with `LV_EVENT_DRAW_TASK_
↪ADDED` event
 */
void lv_example_scale_7(void)
{
    lv_obj_t * scale = lv_scale_create(lv_screen_active());
    lv_obj_set_size(scale, lv_pct(80), 100);
    lv_scale_set_mode(scale, LV_SCALE_MODE_HORIZONTAL_BOTTOM);
    lv_obj_center(scale);

    lv_scale_set_label_show(scale, true);

    lv_scale_set_total_tick_count(scale, 31);
    lv_scale_set_major_tick_every(scale, 5);

    lv_obj_set_style_length(scale, 5, LV_PART_ITEMS);
    lv_obj_set_style_length(scale, 10, LV_PART_INDICATOR);
    lv_scale_set_range(scale, 10, 40);

    lv_obj_add_event_cb(scale, draw_event_cb, LV_EVENT_DRAW_TASK_ADDED,␣
↪NULL);
    lv_obj_add_flag(scale, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);
}

#endif
```

Una scala circolare con etichette ruotate e traslate

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES


/**
 * A simple round scale with label/tick translation
 */
void lv_example_scale_8(void)
{
    lv_obj_t * scale_line = lv_scale_create(lv_screen_active());
    lv_obj_set_size(scale_line, 150, 150);
    lv_scale_set_mode(scale_line, LV_SCALE_MODE_ROUND_INNER);
    lv_obj_set_style_bg_opa(scale_line, LV_OPA_COVER, 0);
    lv_obj_set_style_bg_color(scale_line, lv_palette_lighten(LV_PALETTE_
↪RED, 5), 0);
    lv_obj_set_style_radius(scale_line, LV_RADIUS_CIRCLE, 0);
    lv_obj_align(scale_line, LV_ALIGN_LEFT_MID, LV_PCT(2), 0);

    /*Set the texts' and major ticks' style (make the texts rotated)*/
    lv_obj_set_style_transform_rotation(scale_line, LV_SCALE_LABEL_ROTATE_
↪MATCH_TICKS | LV_SCALE_LABEL_ROTATE_KEEP_UPRIGHT,
                                        LV_PART_INDICATOR);
    lv_obj_set_style_translate_x(scale_line, 10, LV_PART_INDICATOR);
    lv_obj_set_style_length(scale_line, 15, LV_PART_INDICATOR);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_style_radial_offset(scale_line, 10, LV_PART_INDICATOR);

    /*Set the style of the minor ticks*/
    lv_obj_set_style_length(scale_line, 10, LV_PART_ITEMS);
    lv_obj_set_style_radial_offset(scale_line, 5, LV_PART_ITEMS);
    lv_obj_set_style_line_opa(scale_line, LV_OPA_50, LV_PART_ITEMS);

    lv_scale_set_label_show(scale_line, true);

    lv_scale_set_total_tick_count(scale_line, 31);
    lv_scale_set_major_tick_every(scale_line, 5);

    lv_scale_set_range(scale_line, 10, 40);

    lv_scale_set_angle_range(scale_line, 270);
    lv_scale_set_rotation(scale_line, 135);

    lv_obj_t * needle_line = lv_line_create(scale_line);

    lv_obj_set_style_line_width(needle_line, 3, LV_PART_MAIN);
    lv_obj_set_style_line_rounded(needle_line, true, LV_PART_MAIN);
    lv_scale_set_line_needle_value(scale_line, needle_line, 60, 33);
}

#endif
```

Una scala orizzontale con etichette ruotate e traslate

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES

/**
 * A simple horizontal scale with transforms
 */
void lv_example_scale_9(void)
{
    lv_obj_t * scale = lv_scale_create(lv_screen_active());
    lv_obj_set_size(scale, 200, 100);
    lv_scale_set_mode(scale, LV_SCALE_MODE_HORIZONTAL_BOTTOM);
    lv_obj_center(scale);

    lv_scale_set_label_show(scale, true);
    lv_obj_set_style_transform_rotation(scale, 450, LV_PART_INDICATOR);
    lv_obj_set_style_length(scale, 30, LV_PART_INDICATOR);
    lv_obj_set_style_translate_x(scale, 5, LV_PART_INDICATOR);

    lv_scale_set_total_tick_count(scale, 31);
    lv_scale_set_major_tick_every(scale, 5);

    lv_obj_set_style_length(scale, 5, LV_PART_ITEMS);
    lv_obj_set_style_length(scale, 10, LV_PART_INDICATOR);
    lv_scale_set_range(scale, 10, 40);
}

#endif
```

Uno stile di scala circolare che simula un cardiofrequenzimetro

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES && LV_FONT_MONTSERRAT_40 && LV_FONT_
↪MONTSERRAT_18

static int32_t hr_value = 98;
static int8_t hr_step = 1;
static lv_obj_t * needle_line = NULL;
static lv_obj_t * hr_value_label = NULL;
static lv_obj_t * bpm_label = NULL;
static lv_obj_t * scale = NULL;

typedef struct {
    lv_style_t items;
    lv_style_t indicator;
    lv_style_t main;
} section_styles_t;

static section_styles_t zone1_styles;
static section_styles_t zone2_styles;
static section_styles_t zone3_styles;
static section_styles_t zone4_styles;
static section_styles_t zone5_styles;

static lv_color_t get_hr_zone_color(int32_t hr)
{
    if(hr < 117) return lv_palette_main(LV_PALETTE_GREY); /* Zone 1 */
    else if(hr < 135) return lv_palette_main(LV_PALETTE_BLUE); /* Zone 2 */
    else if(hr < 158) return lv_palette_main(LV_PALETTE_GREEN); /* Zone 3␣
↪*/
    else if(hr < 176) return lv_palette_main(LV_PALETTE_ORANGE); /* Zone 4␣
↪*/
    else return lv_palette_main(LV_PALETTE_RED); /* Zone 5 */
}

static void hr_anim_timer_cb(lv_timer_t * timer)
{
    LV_UNUSED(timer);

    hr_value += hr_step;

    if(hr_value >= 195) {
        hr_value = 195;
        hr_step = -1;
    }
    else if(hr_value <= 98) {
        hr_value = 98;
        hr_step = 1;
    }

    /* Update needle */
    lv_scale_set_line_needle_value(scale, needle_line, -8, hr_value);

    /* Update HR text */
    lv_label_set_text_fmt(hr_value_label, "%d", hr_value);
```

(continues on next page)

(continua dalla pagina precedente)

```
    /* Update text color based on zone */
    lv_color_t zone_color = get_hr_zone_color(hr_value);
    lv_obj_set_style_text_color(hr_value_label, zone_color, 0);
    lv_obj_set_style_text_color(bpm_label, zone_color, 0);
}

static void init_section_styles(section_styles_t * styles, lv_color_t␣
↪color)
{
    lv_style_init(&styles->items);
    lv_style_set_line_color(&styles->items, color);
    lv_style_set_line_width(&styles->items, 0);

    lv_style_init(&styles->indicator);
    lv_style_set_line_color(&styles->indicator, color);
    lv_style_set_line_width(&styles->indicator, 0);

    lv_style_init(&styles->main);
    lv_style_set_arc_color(&styles->main, color);
    lv_style_set_arc_width(&styles->main, 20);
}

static void add_section(lv_obj_t * target_scale,
                        int32_t from,
                        int32_t to,
                        const section_styles_t * styles)
{
    lv_scale_section_t * sec = lv_scale_add_section(target_scale);
    lv_scale_set_section_range(target_scale, sec, from, to);
    lv_scale_set_section_style_items(target_scale, sec, &styles->items);
    lv_scale_set_section_style_indicator(target_scale, sec, &styles->
↪indicator);
    lv_scale_set_section_style_main(target_scale, sec, &styles->main);
}

void lv_example_scale_10(void)
{
    scale = lv_scale_create(lv_screen_active());
    lv_obj_center(scale);
    lv_obj_set_size(scale, 200, 200);

    lv_scale_set_mode(scale, LV_SCALE_MODE_ROUND_INNER);
    lv_scale_set_range(scale, 98, 195);
    lv_scale_set_total_tick_count(scale, 15);
    lv_scale_set_major_tick_every(scale, 3);
    lv_scale_set_angle_range(scale, 280);
    lv_scale_set_rotation(scale, 130);
    lv_scale_set_label_show(scale, false);

    lv_obj_set_style_length(scale, 6, LV_PART_ITEMS);
    lv_obj_set_style_length(scale, 10, LV_PART_INDICATOR);
    lv_obj_set_style_arc_width(scale, 0, LV_PART_MAIN);

    /* Zone 1: (Grey) */
    init_section_styles(&zone1_styles, lv_palette_main(LV_PALETTE_GREY));
    add_section(scale, 98, 117, &zone1_styles);
```

(continues on next page)

(continua dalla pagina precedente)

```
    /* Zone 2: (Blue) */
    init_section_styles(&zone2_styles, lv_palette_main(LV_PALETTE_BLUE));
    add_section(scale, 117, 135, &zone2_styles);

    /* Zone 3: (Green) */
    init_section_styles(&zone3_styles, lv_palette_main(LV_PALETTE_GREEN));
    add_section(scale, 135, 158, &zone3_styles);

    /* Zone 4: (Orange) */
    init_section_styles(&zone4_styles, lv_palette_main(LV_PALETTE_ORANGE));
    add_section(scale, 158, 176, &zone4_styles);

    /* Zone 5: (Red) */
    init_section_styles(&zone5_styles, lv_palette_main(LV_PALETTE_RED));
    add_section(scale, 176, 195, &zone5_styles);

    needle_line = lv_line_create(scale);

    /* Optional styling */
    lv_obj_set_style_line_color(needle_line, lv_color_black(), LV_PART_
↪MAIN);
    lv_obj_set_style_line_width(needle_line, 12, LV_PART_MAIN);
    lv_obj_set_style_length(needle_line, 20, LV_PART_MAIN);
    lv_obj_set_style_line_rounded(needle_line, false, LV_PART_MAIN);
    lv_obj_set_style_pad_right(needle_line, 50, LV_PART_MAIN);

    int32_t current_hr = 145;

    lv_scale_set_line_needle_value(scale, needle_line, 0, current_hr);

    lv_obj_t * circle = lv_obj_create(lv_scr_act());
    lv_obj_set_size(circle, 130, 130);
    lv_obj_center(circle);

    lv_obj_set_style_radius(circle, LV_RADIUS_CIRCLE, 0);

    lv_obj_set_style_bg_color(circle, lv_obj_get_style_bg_color(lv_scr_
↪act(), LV_PART_MAIN), 0);
    lv_obj_set_style_bg_opa(circle, LV_OPA_COVER, 0);
    lv_obj_set_style_border_width(circle, 0, LV_PART_MAIN);

    lv_obj_t * hr_container = lv_obj_create(circle);
    lv_obj_center(hr_container);
    lv_obj_set_size(hr_container, lv_pct(100), LV_SIZE_CONTENT);
    lv_obj_set_style_bg_opa(hr_container, LV_OPA_TRANSP, 0);
    lv_obj_set_style_border_width(hr_container, 0, 0);
    lv_obj_set_layout(hr_container, LV_LAYOUT_FLEX);
    lv_obj_set_flex_flow(hr_container, LV_FLEX_FLOW_COLUMN);
    lv_obj_set_style_pad_all(hr_container, 0, LV_PART_MAIN);
    lv_obj_set_style_pad_row(hr_container, 0, 0);
    lv_obj_set_flex_align(hr_container, LV_FLEX_ALIGN_CENTER, LV_FLEX_
↪ALIGN_CENTER, LV_FLEX_ALIGN_CENTER);

    hr_value_label = lv_label_create(hr_container);
    lv_label_set_text_fmt(hr_value_label, "%d", current_hr);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_style_text_font(hr_value_label, &lv_font_montserrat_40, 0);
    lv_obj_set_style_text_align(hr_value_label, LV_TEXT_ALIGN_CENTER, 0);

    bpm_label = lv_label_create(hr_container);
    lv_label_set_text(bpm_label, "bpm");
    lv_obj_set_style_text_font(bpm_label, &lv_font_montserrat_18, 0);
    lv_obj_set_style_text_align(bpm_label, LV_TEXT_ALIGN_CENTER, 0);

    lv_color_t zone_color = get_hr_zone_color(current_hr);
    lv_obj_set_style_text_color(hr_value_label, zone_color, 0);
    lv_obj_set_style_text_color(bpm_label, zone_color, 0);

    lv_timer_create(hr_anim_timer_cb, 80, NULL);
}

#endif
```

Uno stile di scala circolare che simula un widget tramonto/alba

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES && LV_FONT_MONTSERRAT_12 && LV_FONT_
↪MONTSERRAT_14 && LV_FONT_MONTSERRAT_16 && LV_FONT_MONTSERRAT_20

static void label_color_cb(lv_event_t * e)
{
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    if(!draw_task) return;

    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);
    if(!base_dsc || base_dsc->part != LV_PART_INDICATOR) return;

    lv_draw_label_dsc_t * label_dsc = lv_draw_task_get_label_dsc(draw_
↪task);
    if(!label_dsc || !label_dsc->text) return;

    const char * txt = label_dsc->text;

    if(lv_strcmp(txt, "06") == 0 || lv_strcmp(txt, "12") == 0 ||
       lv_strcmp(txt, "18") == 0 || lv_strcmp(txt, "24") == 0) {
        label_dsc->color = lv_color_white();
    }
    else {
        label_dsc->color = lv_palette_darken(LV_PALETTE_GREY, 1);
    }
}

void lv_example_scale_11(void)
{
    lv_obj_t * bg = lv_obj_create(lv_screen_active());
    lv_obj_set_size(bg, 210, 210);
    lv_obj_center(bg);
    lv_obj_set_style_radius(bg, LV_RADIUS_CIRCLE, 0);
    lv_obj_set_style_bg_color(bg, lv_palette_darken(LV_PALETTE_GREY, 4),␣
↪0);
    lv_obj_set_style_bg_opa(bg, LV_OPA_COVER, 0);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_remove_flag(bg, LV_OBJ_FLAG_SCROLLABLE);
    lv_obj_set_style_pad_all(bg, 0, LV_PART_MAIN);

    lv_obj_t * scale = lv_scale_create(bg);
    lv_obj_center(scale);
    lv_obj_set_size(scale, 150, 150);
    lv_obj_set_style_arc_width(scale, 5, LV_PART_MAIN);

    lv_scale_set_mode(scale, LV_SCALE_MODE_ROUND_OUTER);
    lv_scale_set_range(scale, 0, 24);
    lv_scale_set_total_tick_count(scale, 25);
    lv_scale_set_major_tick_every(scale, 1);
    lv_scale_set_angle_range(scale, 360);
    lv_scale_set_rotation(scale, 105);
    lv_scale_set_label_show(scale, true);
    lv_obj_set_style_text_font(scale, &lv_font_montserrat_12, LV_PART_
↪INDICATOR);
    lv_obj_set_style_pad_radial(scale, -6, LV_PART_INDICATOR);

    /*Rotate the labels of the ticks*/
    lv_obj_set_style_transform_rotation(scale, LV_SCALE_LABEL_ROTATE_MATCH_
↪TICKS | LV_SCALE_LABEL_ROTATE_KEEP_UPRIGHT,
                                        LV_PART_INDICATOR);

    /* Style for major ticks */
    static lv_style_t style_ticks;
    lv_style_init(&style_ticks);
    lv_style_set_line_color(&style_ticks, lv_palette_darken(LV_PALETTE_
↪GREY, 1));
    lv_style_set_line_width(&style_ticks, 2);
    lv_style_set_width(&style_ticks, 10);
    lv_obj_add_style(scale, &style_ticks, LV_PART_INDICATOR);

    /* Style for NIGHT - blue */
    static lv_style_t style_night;
    lv_style_init(&style_night);
    lv_style_set_arc_color(&style_night, lv_palette_main(LV_PALETTE_BLUE));

    /* Style for DAY - dark yellow */
    static lv_style_t style_day;
    lv_style_init(&style_day);
    lv_style_set_arc_color(&style_day, lv_palette_darken(LV_PALETTE_YELLOW,
↪ 3));

    /* NIGHT section */
    lv_scale_section_t * section_night1 = lv_scale_add_section(scale);
    lv_scale_set_section_range(scale, section_night1, 17, 5);
    lv_scale_set_section_style_main(scale, section_night1, &style_night);

    /* DAY section */
    lv_scale_section_t * section_day = lv_scale_add_section(scale);
    lv_scale_set_section_range(scale, section_day, 5, 17);
    lv_scale_set_section_style_main(scale, section_day, &style_day);


    static const char * hour_labels[] = {
```

(continues on next page)

(continua dalla pagina precedente)

```
        "01", "02", "03", "04", "05",
        "06", "07", "08", "09", "10",
        "11", "12", "13", "14", "15",
        "16", "17", "18", "19", "20",
        "21", "22", "23", "24",
        NULL
    };
    lv_scale_set_text_src(scale, hour_labels);

    lv_obj_add_flag(scale, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);
    lv_obj_add_event_cb(scale, label_color_cb, LV_EVENT_DRAW_TASK_ADDED,␣
↪NULL);

    lv_obj_t * today = lv_label_create(bg);
    lv_label_set_text(today, "TODAY");
    lv_obj_set_style_text_font(today, &lv_font_montserrat_16, 0);
    lv_obj_set_style_text_color(today, lv_color_white(), 0);
    lv_obj_align(today, LV_ALIGN_TOP_MID, 0, 60);

    lv_obj_t * sunrise_lbl = lv_label_create(bg);
    lv_label_set_text(sunrise_lbl, "SUNRISE");
    lv_obj_set_style_text_font(sunrise_lbl, &lv_font_montserrat_14, 0);
    lv_obj_set_style_text_color(sunrise_lbl, lv_palette_main(LV_PALETTE_
↪GREY), 0);
    lv_obj_align(sunrise_lbl, LV_ALIGN_LEFT_MID, 37, -10);

    lv_obj_t * sunrise_time = lv_label_create(bg);
    lv_label_set_text(sunrise_time, "6:43");
    lv_obj_set_style_text_font(sunrise_time, &lv_font_montserrat_20, 0);
    lv_obj_set_style_text_color(sunrise_time, lv_color_white(), 0);
    lv_obj_align_to(sunrise_time, sunrise_lbl, LV_ALIGN_OUT_BOTTOM_MID, 0,␣
↪2);

    lv_obj_t * sunset_lbl = lv_label_create(bg);
    lv_label_set_text(sunset_lbl, "SUNSET");
    lv_obj_set_style_text_font(sunset_lbl, &lv_font_montserrat_14, 0);
    lv_obj_set_style_text_color(sunset_lbl, lv_palette_main(LV_PALETTE_
↪GREY), 0);
    lv_obj_align(sunset_lbl, LV_ALIGN_RIGHT_MID, -37, -10);

    lv_obj_t * sunset_time = lv_label_create(bg);
    lv_label_set_text(sunset_time, "17:37");
    lv_obj_set_style_text_font(sunset_time, &lv_font_montserrat_20, 0);
    lv_obj_set_style_text_color(sunset_time, lv_color_white(), 0);
    lv_obj_align_to(sunset_time, sunset_lbl, LV_ALIGN_OUT_BOTTOM_MID, 0,␣
↪2);
}

#endif
```

Uno stile di scala circolare che simula una bussola

```
#include "../../lv_examples.h"
#if LV_USE_SCALE && LV_BUILD_EXAMPLES

static lv_obj_t * scale;
```

(continues on next page)

(continua dalla pagina precedente)

```
static lv_obj_t * label;

static const char * heading_to_cardinal(int32_t heading)
{
    /* Normalize heading to range [0, 360) */
    while(heading < 0) heading += 360;
    while(heading >= 360) heading -= 360;

    if(heading < 23) return "N";
    else if(heading < 68) return "NE";
    else if(heading < 113) return "E";
    else if(heading < 158) return "SE";
    else if(heading < 203) return "S";
    else if(heading < 248) return "SW";
    else if(heading < 293) return "W";
    else if(heading < 338) return "NW";

    return "N";
}

static void set_heading_value(void * obj, int32_t v)
{
    LV_UNUSED(obj);
    lv_scale_set_rotation(scale, 270 - v);
    lv_label_set_text_fmt(label, "%d°\n%s", v, heading_to_cardinal(v));
}

static void draw_event_cb(lv_event_t * e)
{
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);
    lv_draw_label_dsc_t * label_draw_dsc = lv_draw_task_get_label_dsc(draw_
↪task);
    lv_draw_line_dsc_t * line_draw_dsc = lv_draw_task_get_line_dsc(draw_
↪task);
    if(base_dsc->part == LV_PART_INDICATOR) {
        if(label_draw_dsc) {
            if(base_dsc->id1 == 0) {
                label_draw_dsc->color = lv_palette_main(LV_PALETTE_RED);
            }
        }
        if(line_draw_dsc) {
            if(base_dsc->id1 == 60) {
                line_draw_dsc->color = lv_palette_main(LV_PALETTE_RED);
            }
        }
    }
}

/**
 * A round scale style simulating a compass
 */
void lv_example_scale_12(void)
{
    scale = lv_scale_create(lv_screen_active());
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_size(scale, 200, 200);
    lv_scale_set_mode(scale, LV_SCALE_MODE_ROUND_INNER);
    lv_obj_set_align(scale, LV_ALIGN_CENTER);

    lv_scale_set_total_tick_count(scale, 61);
    lv_scale_set_major_tick_every(scale, 5);

    lv_obj_set_style_length(scale, 5, LV_PART_ITEMS);
    lv_obj_set_style_length(scale, 10, LV_PART_INDICATOR);
    lv_obj_set_style_line_width(scale, 3, LV_PART_INDICATOR);
    lv_scale_set_range(scale, 0, 360);

    static const char * custom_labels[] = {"N", "30", "60", "E", "120",
↪"150", "S", "210", "240", "W", "300", "330", NULL};
    lv_scale_set_text_src(scale, custom_labels);

    lv_scale_set_angle_range(scale, 360);
    lv_scale_set_rotation(scale, 270);

    lv_obj_add_event_cb(scale, draw_event_cb, LV_EVENT_DRAW_TASK_ADDED,␣
↪NULL);
    lv_obj_add_flag(scale, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);

    label = lv_label_create(lv_screen_active());
    lv_obj_set_width(label, 100);
    lv_obj_set_align(label, LV_ALIGN_CENTER);
    lv_label_set_text(label, "0°\nN");
    lv_obj_set_style_text_align(label, LV_TEXT_ALIGN_CENTER, 0);

    set_heading_value(NULL, 0);

    lv_obj_t * symbol = lv_label_create(scale);
    lv_obj_set_align(symbol, LV_ALIGN_TOP_MID);
    lv_obj_set_y(symbol, 5);
    lv_label_set_text(symbol, LV_SYMBOL_UP);
    lv_obj_set_style_text_align(symbol, LV_TEXT_ALIGN_CENTER, 0);
    lv_obj_set_style_text_color(symbol, lv_palette_main(LV_PALETTE_RED),␣
↪0);

    lv_anim_t anim_scale;
    lv_anim_init(&anim_scale);
    lv_anim_set_var(&anim_scale, scale);
    lv_anim_set_exec_cb(&anim_scale, set_heading_value);
    lv_anim_set_duration(&anim_scale, 5000);
    lv_anim_set_repeat_delay(&anim_scale, 500);
    lv_anim_set_repeat_count(&anim_scale, LV_ANIM_REPEAT_INFINITE);
    lv_anim_set_reverse_duration(&anim_scale, 5000);
    lv_anim_set_reverse_delay(&anim_scale, 500);
    lv_anim_set_values(&anim_scale, 0, 360);
    lv_anim_start(&anim_scale);

}

#endif
```

Segni di spunta e etichette degli assi con scorrimento su un grafico

```
#include "../../lv_examples.h"
#if LV_USE_CHART && LV_BUILD_EXAMPLES

/**
 * Use lv_scale to add ticks to a scrollable chart
 */
void lv_example_chart_2(void)
{
    /*Create a container*/
    lv_obj_t * main_cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(main_cont, 200, 150);
    lv_obj_center(main_cont);

    /*Create a transparent wrapper for the chart and the scale.
     *Set a large width, to make it scrollable on the main container*/
    lv_obj_t * wrapper = lv_obj_create(main_cont);
    lv_obj_remove_style_all(wrapper);
    lv_obj_set_size(wrapper, lv_pct(300), lv_pct(100));
    lv_obj_set_flex_flow(wrapper, LV_FLEX_FLOW_COLUMN);

    /*Create a chart on the wrapper
     *Set it's width to 100% to fill the large wrapper*/
    lv_obj_t * chart = lv_chart_create(wrapper);
    lv_obj_set_width(chart, lv_pct(100));
    lv_obj_set_flex_grow(chart, 1);
    lv_chart_set_type(chart, LV_CHART_TYPE_BAR);
    lv_chart_set_axis_range(chart, LV_CHART_AXIS_PRIMARY_Y, 0, 100);
    lv_chart_set_axis_range(chart, LV_CHART_AXIS_SECONDARY_Y, 0, 400);
    lv_chart_set_point_count(chart, 12);
    lv_obj_set_style_radius(chart, 0, 0);

    /*Create a scale also with 100% width*/
    lv_obj_t * scale_bottom = lv_scale_create(wrapper);
    lv_scale_set_mode(scale_bottom, LV_SCALE_MODE_HORIZONTAL_BOTTOM);
    lv_obj_set_size(scale_bottom, lv_pct(100), 25);
    lv_scale_set_total_tick_count(scale_bottom, 12);
    lv_scale_set_major_tick_every(scale_bottom, 1);
    lv_obj_set_style_pad_hor(scale_bottom, lv_chart_get_first_point_center_
↪offset(chart), 0);

    static const char * month[] = {"Jan", "Febr", "March", "Apr", "May",
↪"Jun", "July", "Aug", "Sept", "Oct", "Nov", "Dec", NULL};
    lv_scale_set_text_src(scale_bottom, month);

    /*Add two data series*/
    lv_chart_series_t * ser1 = lv_chart_add_series(chart, lv_palette_
↪lighten(LV_PALETTE_GREEN, 2), LV_CHART_AXIS_PRIMARY_Y);
    lv_chart_series_t * ser2 = lv_chart_add_series(chart, lv_palette_
↪darken(LV_PALETTE_GREEN, 2), LV_CHART_AXIS_PRIMARY_Y);

    /*Set the next points on 'ser1'*/
    uint32_t i;
    for(i = 0; i < 12; i++) {
        lv_chart_set_next_value(chart, ser1, (int32_t)lv_rand(10, 60));
        lv_chart_set_next_value(chart, ser2, (int32_t)lv_rand(50, 90));
```

(continues on next page)

(continua dalla pagina precedente)

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

#endif
```

6.25.7 API

Scale (lv_scale) .. Autogenerato

lv_scale.h

lv_scale_private.h

6.26 Slider (lv_slider)

6.26.1 Panoramica

The Slider Widget looks like a *Bar (lv_bar)* supplemented with a knob. The knob can be dragged to set the Slider's value. Like Bar, a Slider can be vertical or horizontal.

6.26.2 Parts and Styles

- `LV_PART_MAIN` The background of the Slider. Uses the *typical background style properties*. `padding` makes the indicator smaller in the respective direction.
- `LV_PART_INDICATOR` The indicator that shows the current state of the Slider; also uses the *typical background style properties*.
- `LV_PART_KNOB` A rectangle (or circle) drawn at the current value; also uses the *typical background style properties* to describe the knob(s). By default, the knob is round (radius-style can modify this) with side length equal to the smaller dimension of the Slider. The knob can be made larger with the `padding` values. Padding values can be asymmetric as well.

6.26.3 Utilizzo

Value, range and orientation

Once a Slider is created, it has:

- value == 0
- default range of [0..100],
- horizontal orientation, with
- default width of approximately 2 inches (according to configured value of `LV_DPI_DEF`),
- default height of approximately 1/10 inch (according to configured value of `LV_DPI_DEF`).

To set different values use:

- `lv_slider_set_value(slider, new_value, LV_ANIM_ON / OFF)` (animation time is set by the styles' `anim_time` property);
- `lv_slider_set_range(slider, min, max)`; and
- for orientation, width and height, simply set width and height properties;
- `lv_slider_set_orientation(slider, orientation)` to override orientation caused by `width` and `height`. Valid values for `orientation` are:
  - `LV_SLIDER_ORIENTATION_AUTO`

  - *LV_SLIDER_ORIENTATION_HORIZONTAL*
  - *LV_SLIDER_ORIENTATION_VERTICAL*

The default drawing direction is from left to right in horizontal orientation and bottom to top in vertical orientation. If the minimum value is set to be greater than the maximum value (e.g. [100..0]), the drawing direction is reversed.

Modes

The Slider can be in one of the following modes:

- *LV_SLIDER_MODE_NORMAL* A normal Slider as described above (default)
- `LV_SLIDER_SYMMETRICAL` Draw the indicator from the zero value to current value. Requires negative minimum range and positive maximum range.
- `LV_SLIDER_RANGE` Allows setting the start value as well by *lv_bar_set_start_value*`(slider, new_value, LV_ANIM_ON / OFF)`. The start value must always be smaller than the end value.

The mode can be changed with *lv_slider_set_mode*`(slider, LV_SLIDER_MODE_...)`

Knob-only mode

Normally, the Slider can be adjusted either by dragging the knob, or by clicking on the Slider bar. In the latter case the knob moves to the point clicked and the Slider value changes accordingly. In some cases it is desirable to set the Slider to react on dragging the knob only. This feature is enabled by adding the *LV_OBJ_FLAG_ADV_HITTEST* flag: *lv_obj_add_flag*`(slider, LV_OBJ_FLAG_ADV_HITTEST)`.

Any extended click area (set by *lv_obj_set_ext_click_area*`(slider, value)`) increases the knob's click area.

6.26.4 Eventi

- *LV_EVENT_VALUE_CHANGED* Sent while the Slider is being dragged or changed with keys. The event is sent continuously while the Slider is being dragged.
- *LV_EVENT_RELEASED* Sent once when Slider is released.

Further Reading

Bar Events.

Learn more about *Base-Widget Events* emitted by all Widgets.

Learn more about *Eventi*.

6.26.5 Keys

- `LV_KEY_UP/RIGHT` Increment Slider's value by 1.
- `LV_KEY_DOWN/LEFT` Decrement Slider's value by 1.

Further Reading

Learn more about *Keys*.

6.26.6 Esempio

Slider semplice

```
#include "../../lv_examples.h"
#if LV_USE_SLIDER && LV_BUILD_EXAMPLES

static void slider_event_cb(lv_event_t * e);
static lv_obj_t * slider_label;

/**
 * A default slider with a label displaying the current value
 */
void lv_example_slider_1(void)
{
    /*Create a slider in the center of the display*/
    lv_obj_t * slider = lv_slider_create(lv_screen_active());
    lv_obj_center(slider);
    lv_obj_add_event_cb(slider, slider_event_cb, LV_EVENT_VALUE_CHANGED,␣
↪NULL);

    lv_obj_set_style_anim_duration(slider, 2000, 0);
    /*Create a label below the slider*/
    slider_label = lv_label_create(lv_screen_active());
    lv_label_set_text(slider_label, "0%");

    lv_obj_align_to(slider_label, slider, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
}

static void slider_event_cb(lv_event_t * e)
{
    lv_obj_t * slider = lv_event_get_target_obj(e);
    char buf[8];
    lv_snprintf(buf, sizeof(buf), "%d%%", (int)lv_slider_get_
↪value(slider));
    lv_label_set_text(slider_label, buf);
    lv_obj_align_to(slider_label, slider, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
}

#endif
```

Slider con stile personalizzato

```
#include "../../lv_examples.h"
#if LV_USE_SLIDER && LV_BUILD_EXAMPLES

/**
 * Show how to style a slider.
 */
void lv_example_slider_2(void)
{
    /*Create a transition*/
    static const lv_style_prop_t props[] = {LV_STYLE_BG_COLOR, 0};
    static lv_style_transition_dsc_t transition_dsc;
    lv_style_transition_dsc_init(&transition_dsc, props, lv_anim_path_
↪linear, 300, 0, NULL);

    static lv_style_t style_main;
    static lv_style_t style_indicator;
    static lv_style_t style_knob;
```

(continues on next page)

(continua dalla pagina precedente)

```
    static lv_style_t style_pressed_color;
    lv_style_init(&style_main);
    lv_style_set_bg_opa(&style_main, LV_OPA_COVER);
    lv_style_set_bg_color(&style_main, lv_color_hex3(0xbbb));
    lv_style_set_radius(&style_main, LV_RADIUS_CIRCLE);
    lv_style_set_pad_ver(&style_main, -2); /*Makes the indicator larger*/

    lv_style_init(&style_indicator);
    lv_style_set_bg_opa(&style_indicator, LV_OPA_COVER);
    lv_style_set_bg_color(&style_indicator, lv_palette_main(LV_PALETTE_
↪CYAN));
    lv_style_set_radius(&style_indicator, LV_RADIUS_CIRCLE);
    lv_style_set_transition(&style_indicator, &transition_dsc);

    lv_style_init(&style_knob);
    lv_style_set_bg_opa(&style_knob, LV_OPA_COVER);
    lv_style_set_bg_color(&style_knob, lv_palette_main(LV_PALETTE_CYAN));
    lv_style_set_border_color(&style_knob, lv_palette_darken(LV_PALETTE_
↪CYAN, 3));
    lv_style_set_border_width(&style_knob, 2);
    lv_style_set_radius(&style_knob, LV_RADIUS_CIRCLE);
    lv_style_set_pad_all(&style_knob, 6); /*Makes the knob larger*/
    lv_style_set_transition(&style_knob, &transition_dsc);

    lv_style_init(&style_pressed_color);
    lv_style_set_bg_color(&style_pressed_color, lv_palette_darken(LV_
↪PALETTE_CYAN, 2));

    /*Create a slider and add the style*/
    lv_obj_t * slider = lv_slider_create(lv_screen_active());
    lv_obj_remove_style_all(slider);        /*Remove the styles coming␣
↪from the theme*/

    lv_obj_add_style(slider, &style_main, LV_PART_MAIN);
    lv_obj_add_style(slider, &style_indicator, LV_PART_INDICATOR);
    lv_obj_add_style(slider, &style_pressed_color, LV_PART_INDICATOR | LV_
↪STATE_PRESSED);
    lv_obj_add_style(slider, &style_knob, LV_PART_KNOB);
    lv_obj_add_style(slider, &style_pressed_color, LV_PART_KNOB | LV_STATE_
↪PRESSED);

    lv_obj_center(slider);
}

#endif
```

Slider estensibile

```
#include "../../lv_examples.h"
#if LV_USE_SLIDER && LV_BUILD_EXAMPLES

#define MAX_VALUE 100
#define MIN_VALUE 0

static void slider_event_cb(lv_event_t * e);
```

(continues on next page)

(continua dalla pagina precedente)

```
/**
 * Show the current value when the slider is pressed by extending the␣
↪drawer
 *
 */
void lv_example_slider_3(void)
{
    /*Create a slider in the center of the display*/
    lv_obj_t * slider;
    slider = lv_slider_create(lv_screen_active());
    lv_obj_center(slider);

    lv_slider_set_mode(slider, LV_SLIDER_MODE_RANGE);
    lv_slider_set_range(slider, MIN_VALUE, MAX_VALUE);
    lv_slider_set_value(slider, 70, LV_ANIM_OFF);
    lv_slider_set_start_value(slider, 20, LV_ANIM_OFF);

    lv_obj_add_event_cb(slider, slider_event_cb, LV_EVENT_ALL, NULL);
    lv_obj_refresh_ext_draw_size(slider);
}

static void slider_event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);

    /*Provide some extra space for the value*/
    if(code == LV_EVENT_REFR_EXT_DRAW_SIZE) {
        lv_event_set_ext_draw_size(e, 50);
    }
    else if(code == LV_EVENT_DRAW_MAIN_END) {
        if(!lv_obj_has_state(obj, LV_STATE_PRESSED)) return;

        lv_area_t slider_area;
        lv_obj_get_coords(obj, &slider_area);
        lv_area_t indic_area = slider_area;
        lv_area_set_width(&indic_area, lv_area_get_width(&slider_area) *␣
↪lv_slider_get_value(obj) / MAX_VALUE);
        indic_area.x1 += lv_area_get_width(&slider_area) * lv_slider_get_
↪left_value(obj) / MAX_VALUE;
        char buf[16];
        lv_snprintf(buf, sizeof(buf), "%d - %d", (int)lv_slider_get_left_
↪value(obj), (int)lv_slider_get_value(obj));

        lv_point_t label_size;
        lv_text_get_size(&label_size, buf, LV_FONT_DEFAULT, 0, 0, LV_COORD_
↪MAX, LV_TEXT_FLAG_NONE);
        lv_area_t label_area;
        label_area.x1 = 0;
        label_area.x2 = label_size.x - 1;
        label_area.y1 = 0;
        label_area.y2 = label_size.y - 1;

        lv_area_align(&indic_area, &label_area, LV_ALIGN_OUT_TOP_MID, 0, -
↪10);
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_draw_label_dsc_t label_draw_dsc;
        lv_draw_label_dsc_init(&label_draw_dsc);
        label_draw_dsc.color = lv_color_hex3(0x888);
        label_draw_dsc.text = buf;
        label_draw_dsc.text_local = true;
        lv_layer_t * layer = lv_event_get_layer(e);
        lv_draw_label(layer, &label_draw_dsc, &label_area);
    }
}

#endif
```

Slider con direzione opposta

```
#include "../../lv_examples.h"
#if LV_USE_SLIDER && LV_BUILD_EXAMPLES

static void slider_event_cb(lv_event_t * e);
static lv_obj_t * slider_label;

/**
 * Slider with opposite direction
 */
void lv_example_slider_4(void)
{
    /*Create a slider in the center of the display*/
    lv_obj_t * slider = lv_slider_create(lv_screen_active());
    lv_obj_center(slider);
    lv_obj_add_event_cb(slider, slider_event_cb, LV_EVENT_VALUE_CHANGED,␣
↪NULL);
    /*Reverse the direction of the slider*/
    lv_slider_set_range(slider, 100, 0);
    /*Create a label below the slider*/
    slider_label = lv_label_create(lv_screen_active());
    lv_label_set_text(slider_label, "0%");

    lv_obj_align_to(slider_label, slider, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
}

static void slider_event_cb(lv_event_t * e)
{
    lv_obj_t * slider = lv_event_get_target_obj(e);
    char buf[8];
    lv_snprintf(buf, sizeof(buf), "%d%%", (int)lv_slider_get_
↪value(slider));
    lv_label_set_text(slider_label, buf);
    lv_obj_align_to(slider_label, slider, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
}

#endif
```

6.26.7 API

lv_api_map_v9_1.h

lv_obj_property_names.h

lv_observer.h

lv_slider.h

lv_slider_private.h

6.27 Spangroup (lv_spangroup)

6.27.1 Panoramica

The Spangroup Widget is used to display rich text. Different from the Label Widget, Spangroups can render text styled with different fonts, colors, and sizes into the Spangroup Widget. See example below.

A Spangroup contains 0 or more Span Descriptors ("Spans"). Each Span contains its own text and style properties for that text. You add 1 Span (as a child) to the Spangroup for each "span" of uniquely-styled text needed. Each Span so added is appended to the end of the list. The list sequence determines the order in which the Spans are displayed. Spans can be added to, and removed from, the Spangroup throughout its life. The number of Spans that can be added is limited only by available RAM.

6.27.2 Parts and Styles

- *LV_PART_MAIN* Spangroup has only one part.

6.27.3 Utilizzo

Set text and style

Add each needed Span to a Spangroup like this:

```
lv_span_t * span = lv_spangroup_new_span(spangroup);
```

After a Span is created, use the following functions to set its text and style properties:

- *lv_span_set_text*(span, "text")
- lv_style_set_<property_name>(&span->style, value)

Example of the latter: *lv_style_set_text_color*(&span->style, *lv_palette_main*(*LV_PALETTE_RED*)).

If the Spangroup Widget's `mode != LV_SPAN_MODE_FIXED` call `lv_spangroup_refr_mode(spangroup)` after you have modifying any of its Spans to ensure it is redrawn appropriately.

Retrieving a Span child

Spangroups store their children differently from normal Widgets, so normal functions for getting children won't work.

lv_spangroup_get_child(spangroup, id) will return a pointer to the child Span at index `id`. In addition, `id` can be negative to index from the end of the Spangroup where `-1` is the youngest child, `-2` is second youngest, etc.

E.g. `lv_span_t * span = lv_spangroup_get_child(spangroup, 0)` will return the first child of the Spangroup. `lv_span_t * span = lv_spangroup_get_child(spangroup, -1)` will return the last (or most recent) child.

Child count

Use `lv_spangroup_get_span_count(spangroup)` to get the number of contained Spans.

E.g. `uint32_t size = lv_spangroup_get_span_count(spangroup)`

Removing a Span

You can remove a Span at any time during the Spangroup's life using the function `lv_spangroup_delete_span(spangroup, span)`.

Text align

Like the Label Widget, a Spangroup can be set to one the following text-alignment modes:

- `LV_TEXT_ALIGN_LEFT` Align text to left.
- `LV_TEXT_ALIGN_CENTER` Center text.
- `LV_TEXT_ALIGN_RIGHT` Align text to right edge.
- `LV_TEXT_ALIGN_AUTO` Align auto.

Use function `lv_spangroup_set_align(spangroup, LV_TEXT_ALIGN_...)` to set text alignment.

Modes

A Spangroup can be set to one the following modes:

- `LV_SPAN_MODE_FIXED` Fixes its size.
- `LV_SPAN_MODE_EXPAND` Expand size to text size but stay on one line.
- `LV_SPAN_MODE_BREAK` Keep width; break lines that are too long and auto-expand height.

Use `lv_spangroup_set_mode(spangroup, LV_SPAN_MODE_BREAK)` to set its mode.

Overflow

A Spangroup can be set to handle text overflow in one of the following ways:

- `LV_SPAN_OVERFLOW_CLIP` truncates text at the limit of the area.
- `LV_SPAN_OVERFLOW_ELLIPSIS` display an ellipsis (`...`) when text overflows the area.

Use `lv_spangroup_set_overflow(spangroup, LV_SPAN_OVERFLOW_CLIP)` to set the Spangroup's overflow mode.

First line indent

Use `lv_spangroup_set_indent(spangroup, 20)` to set the indent of the first line. All modes support pixel units. In addition, `LV_SPAN_MODE_FIXED` and `LV_SPAN_MODE_BREAK` modes support *percentage units*. as well.

Lines

Use `lv_spangroup_set_max_lines(spangroup, 10)` to set the maximum number of lines to be displayed in `LV_SPAN_MODE_BREAK` mode. A negative value indicates no limit.

6.27.4 Eventi

No special events are sent by Span Widgets.

Further Reading

Learn more about *Base-Widget Events* emitted by all Widgets.

Learn more about *Eventi*.

6.27.5 Keys

No *Keys* are processed by Span Widgets.

Further Reading

Learn more about *Keys*.

6.27.6 Esempio

Span con stili personalizzati

```
#include "../../lv_examples.h"
#if LV_USE_SPAN && LV_BUILD_EXAMPLES

static void click_event_cb(lv_event_t * e)
{
    lv_indev_t * indev = lv_event_get_indev(e);
    lv_point_t point;
    lv_indev_get_point(indev, &point);
#if LV_USE_LOG
    lv_obj_t * spans = lv_event_get_target_obj(e);
    lv_span_t * span = lv_spangroup_get_span_by_point(spans, &point);
    LV_LOG_USER("%s", span ? lv_span_get_text(span) : "NULL");
#endif
}

/**
 * Create spans and get clicked one
 */
void lv_example_span_1(void)
{
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_border_width(&style, 1);
    lv_style_set_border_color(&style, lv_palette_main(LV_PALETTE_ORANGE));
    lv_style_set_pad_all(&style, 2);

    lv_obj_t * spans = lv_spangroup_create(lv_screen_active());
    /* Setting a fixed width and height to LV_SIZE_CONTENT will make the␣
↪text wrap */
    lv_obj_set_width(spans, 300);
    lv_obj_set_height(spans, LV_SIZE_CONTENT);
    lv_obj_center(spans);
    lv_obj_add_style(spans, &style, 0);
    lv_obj_add_flag(spans, LV_OBJ_FLAG_CLICKABLE);

    lv_spangroup_set_align(spans, LV_TEXT_ALIGN_LEFT);
    lv_spangroup_set_overflow(spans, LV_SPAN_OVERFLOW_CLIP);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_spangroup_set_indent(spans, 20);

    lv_span_t * span = lv_spangroup_add_span(spans);
    lv_span_set_text(span, "China is a beautiful country.");
    lv_style_set_text_color(lv_span_get_style(span), lv_palette_main(LV_
↪PALETTE_RED));
    lv_style_set_text_decor(lv_span_get_style(span), LV_TEXT_DECOR_
↪UNDERLINE);
    lv_style_set_text_opa(lv_span_get_style(span), LV_OPA_50);

    span = lv_spangroup_add_span(spans);
    lv_span_set_text_static(span, "good good study, day day up.");
#if LV_FONT_MONTSERRAT_24
    lv_style_set_text_font(lv_span_get_style(span),  &lv_font_montserrat_
↪24);
#endif
    lv_style_set_text_color(lv_span_get_style(span), lv_palette_main(LV_
↪PALETTE_GREEN));

    span = lv_spangroup_add_span(spans);
    lv_span_set_text_static(span, "LVGL is an open-source graphics library.
↪");
    lv_style_set_text_color(lv_span_get_style(span), lv_palette_main(LV_
↪PALETTE_BLUE));

    span = lv_spangroup_add_span(spans);
    lv_span_set_text_static(span, "the boy no name.");
    lv_style_set_text_color(lv_span_get_style(span), lv_palette_main(LV_
↪PALETTE_GREEN));
#if LV_FONT_MONTSERRAT_20
    lv_style_set_text_font(lv_span_get_style(span), &lv_font_montserrat_
↪20);
#endif
    lv_style_set_text_decor(lv_span_get_style(span), LV_TEXT_DECOR_
↪UNDERLINE);

    span = lv_spangroup_add_span(spans);
    lv_span_set_text(span, "I have a dream that hope to come true.");
    lv_style_set_text_decor(lv_span_get_style(span), LV_TEXT_DECOR_
↪STRIKETHROUGH);

    lv_spangroup_refresh(spans);

    lv_obj_add_event_cb(spans, click_event_cb, LV_EVENT_CLICKED, NULL);
}

#endif
```

6.27.7 API

lv_api_map_v9_1.h

lv_span.h

lv_span_private.h

6.28 Spinbox (lv_spinbox)

6.28.1 Panoramica

Spinbox contains an integer displayed as a decimal number with a possible fixed decimal point position and a configurable number of digits. The value can be increased or decreased by *Keys* or API functions. Under the hood Spinbox is a *Text Area (lv_textarea)* with behaviors extended to enable a numeric value to be viewed and modified with configurable constraints.

6.28.2 Parts and Styles

Spinbox's parts are identical to those of *Text Area*.

Value, range and step

- `lv_spinbox_set_value(spinbox, 1234)` sets a new value for the Spinbox.
- `lv_spinbox_increment(spinbox)` and `lv_spinbox_decrement(spinbox)` increments/decrements the value of the Spinbox according to the currently-selected digit.
- `lv_spinbox_set_range(spinbox, -1000, 2500)` sets its range. If the value is changed by `lv_spinbox_set_value(spinbox)`, by *Keys*, by `lv_spinbox_increment(spinbox)` or `lv_spinbox_decrement(spinbox)` this range will be respected.
- `lv_spinbox_set_step(spinbox, 100)` sets which digit to change on increment/decrement. Only multiples of ten can be set.
- `lv_spinbox_set_cursor_pos(spinbox, 1)` sets the cursor to a specific digit to change on increment/decrement. Position '0' sets the cursor to the least significant digit.

If an encoder is used as input device, the selected digit is shifted to the right by default whenever the encoder button is clicked. To change this behavior to shifting to the left, the `lv_spinbox_set_digit_step_direction(spinbox, LV_DIR_LEFT)` can be used.

Format

`lv_spinbox_set_digit_format(spinbox, digit_count, separator_position)` sets the number format. `digit_count` is the total number of digits to display. `separator_position` is the number of leading digits before the decimal point. Pass 0 for `separator_position` to display no decimal point.

Rollover

`lv_spinbox_set_rollover(spinbox, true / false)` enables/disables rollover mode. If either the minimum or maximum value is reached with rollover enabled, and the user attempts to continue changing the value in the same direction, the value will change to the other limit. If rollover is disabled the value will stop at the minimum or maximum value.

6.28.3 Eventi

- `LV_EVENT_VALUE_CHANGED` Sent when value has changed.

Further Reading

Textarea Events.

Learn more about *Base-Widget Events* emitted by all Widgets.

Learn more about *Eventi*.

6.28.4 Keys

- `LV_KEY_LEFT/RIGHT` With *Keypad* move the cursor left/right. With *Encoder* decrement/increment the selected digit.
- `LV_KEY_UP/DOWN` With *Keypad* and *Encoder* increment/decrement the value.
- *`LV_KEY_ENTER`* With *Encoder*, move focus to next digit. If focus is on last digit, focus moves to first digit.

Further Reading

Learn more about *Keys*.

6.28.5 Esempio

Spinbox semplice

```
#include "../../lv_examples.h"
#if LV_USE_SPINBOX && LV_BUILD_EXAMPLES

static lv_obj_t * spinbox;

static void lv_spinbox_increment_event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    if(code == LV_EVENT_SHORT_CLICKED || code  == LV_EVENT_LONG_PRESSED_
↪REPEAT) {
        lv_spinbox_increment(spinbox);
    }
}

static void lv_spinbox_decrement_event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    if(code == LV_EVENT_SHORT_CLICKED || code == LV_EVENT_LONG_PRESSED_
↪REPEAT) {
        lv_spinbox_decrement(spinbox);
    }
}

void lv_example_spinbox_1(void)
{
    spinbox = lv_spinbox_create(lv_screen_active());
    lv_spinbox_set_range(spinbox, -1000, 25000);
    lv_spinbox_set_digit_format(spinbox, 5, 2);
    lv_spinbox_step_prev(spinbox);
    lv_obj_set_width(spinbox, 100);
    lv_obj_center(spinbox);

    int32_t h = lv_obj_get_height(spinbox);

    lv_obj_t * btn = lv_button_create(lv_screen_active());
    lv_obj_set_size(btn, h, h);
    lv_obj_align_to(btn, spinbox, LV_ALIGN_OUT_RIGHT_MID, 5, 0);
    lv_obj_set_style_bg_image_src(btn, LV_SYMBOL_PLUS, 0);
    lv_obj_add_event_cb(btn, lv_spinbox_increment_event_cb, LV_EVENT_ALL, ␣
↪NULL);
```

(continues on next page)

(continua dalla pagina precedente)

```
    btn = lv_button_create(lv_screen_active());
    lv_obj_set_size(btn, h, h);
    lv_obj_align_to(btn, spinbox, LV_ALIGN_OUT_LEFT_MID, -5, 0);
    lv_obj_set_style_bg_image_src(btn, LV_SYMBOL_MINUS, 0);
    lv_obj_add_event_cb(btn, lv_spinbox_decrement_event_cb, LV_EVENT_ALL,␣
↪NULL);
}

#endif
```

6.28.6 API

lv_spinbox.h

lv_spinbox_private.h

6.29 Spinner (lv_spinner)

6.29.1 Panoramica

The Spinner Widget is a spinning arc over a ring, typically used to show some type of activity is in progress.

6.29.2 Parts and Styles

Spinner's parts are identical to those of *Arc*.

6.29.3 Utilizzo

Create a spinner

To create a spinner use `lv_spinner_create(parent)`.

Use `lv_spinner_set_anim_params(spinner, spin_duration, angle)` to customize the duration of one revolution and the length of the arc.

6.29.4 Eventi

No special events are sent by Spinner Widgets.

Further Reading

Arc Events

Learn more about *Base-Widget Events* emitted by all Widgets.

Learn more about *Eventi*.

6.29.5 Keys

No *Keys* are processed by Spinner Widgets.

Further Reading

Learn more about *Keys*.

6.29.6 Esempio

Spinner semplice

```
#include "../../lv_examples.h"
#if LV_USE_SPINNER && LV_BUILD_EXAMPLES

void lv_example_spinner_1(void)
{
    /*Create a spinner*/
    lv_obj_t * spinner = lv_spinner_create(lv_screen_active());
    lv_obj_set_size(spinner, 100, 100);
    lv_obj_center(spinner);
    lv_spinner_set_anim_params(spinner, 10000, 200);
}

#endif
```

6.29.7 API

lv_spinner.h

6.30 Switch (lv_switch)

6.30.1 Panoramica

Switch Widgets look like little sliders and are used to display, and optionally modify, a value that can be "on" or "off".

By default, a Switch is oriented horizontally. Its orientation will be vertical if you set `width < height`.

6.30.2 Parts and Styles

- *`LV_PART_MAIN`* Switch's background; uses the *typical background style properties*. `padding` makes the indicator smaller in the respective direction.
- *`LV_PART_INDICATOR`* The indicator that shows the current state of the Switch; also uses the *typical background style properties*.
- *`LV_PART_KNOB`* A rectangle (or circle) drawn at the left or right side of the indicator; also uses the *typical background style properties* to modify the knob's appearance. By default, the knob is round (radius-style can modify this) with diameter slightly smaller than the smaller side of the slider. The knob can be made larger with the `padding` values. Padding values can be asymmetric as well.

6.30.3 Utilizzo

Change state

The Switch uses the standard *`LV_STATE_CHECKED`* state.

To get the current state of the Switch (with `true` being ON), use *`lv_obj_has_state`*`(widget, `*`LV_STATE_CHECKED`*`)`.

Call *`lv_obj_add_state`*`(widget, `*`LV_STATE_CHECKED`*`)` to turn it on, or *`lv_obj_remove_state`*`(widget, `*`LV_STATE_CHECKED`*`)` to turn it off programmatically.

Change orientation

With a Switch is created, its default orientation is *`LV_SWITCH_ORIENTATION_AUTO`*, which causes it to be oriented based on `width` and `height`. You can change this behavior using *`lv_switch_set_orientation`*`(widget, orientation)`. Possible values for `orientation` are:

- *LV_SWITCH_ORIENTATION_AUTO*
- *LV_SWITCH_ORIENTATION_HORIZONTAL*
- *LV_SWITCH_ORIENTATION_VERTICAL*

6.30.4 Eventi

- *LV_EVENT_VALUE_CHANGED* Sent when Switch changes state.

Further Reading

Learn more about *Base-Widget Events* emitted by all Widgets.

Learn more about *Eventi*.

6.30.5 Keys

- LV_KEY_UP/RIGHT Turns Switch ON
- LV_KEY_DOWN/LEFT Turns Switch OFF
- *LV_KEY_ENTER* Toggles the Switch

Further Reading

Learn more about *Keys*.

6.30.6 Esempio

Switch semplice

```
#include "../../lv_examples.h"
#if LV_USE_SWITCH && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        LV_UNUSED(obj);
        LV_LOG_USER("State: %s\n", lv_obj_has_state(obj, LV_STATE_CHECKED)␣
↪? "On" : "Off");
    }
}

void lv_example_switch_1(void)
{
    lv_obj_set_flex_flow(lv_screen_active(), LV_FLEX_FLOW_COLUMN);
    lv_obj_set_flex_align(lv_screen_active(), LV_FLEX_ALIGN_CENTER, LV_
↪FLEX_ALIGN_CENTER, LV_FLEX_ALIGN_CENTER);

    lv_obj_t * sw;

    sw = lv_switch_create(lv_screen_active());
    lv_obj_add_event_cb(sw, event_handler, LV_EVENT_ALL, NULL);
    lv_obj_add_flag(sw, LV_OBJ_FLAG_EVENT_BUBBLE);
```

(continues on next page)

(continua dalla pagina precedente)

```
    sw = lv_switch_create(lv_screen_active());
    lv_obj_add_state(sw, LV_STATE_CHECKED);
    lv_obj_add_event_cb(sw, event_handler, LV_EVENT_ALL, NULL);

    sw = lv_switch_create(lv_screen_active());
    lv_obj_add_state(sw, LV_STATE_DISABLED);
    lv_obj_add_event_cb(sw, event_handler, LV_EVENT_ALL, NULL);

    sw = lv_switch_create(lv_screen_active());
    lv_obj_add_state(sw, LV_STATE_CHECKED);
    lv_obj_add_state(sw, LV_STATE_DISABLED);
    lv_obj_add_event_cb(sw, event_handler, LV_EVENT_ALL, NULL);
}

#endif
```

Orientamento dello Switch

```
#include "../../lv_examples.h"
#if LV_USE_SWITCH && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(code == LV_EVENT_VALUE_CHANGED) {
        LV_UNUSED(obj);
        LV_LOG_USER("State: %s\n", lv_obj_has_state(obj, LV_STATE_CHECKED)␣
↪? "On" : "Off");
    }
}

void lv_example_switch_2(void)
{
    lv_obj_set_flex_flow(lv_screen_active(), LV_FLEX_FLOW_ROW);
    lv_obj_set_flex_align(lv_screen_active(), LV_FLEX_ALIGN_CENTER, LV_
↪FLEX_ALIGN_CENTER, LV_FLEX_ALIGN_CENTER);

    lv_obj_t * sw;

    sw = lv_switch_create(lv_screen_active());
    lv_obj_set_size(sw, 30, 60);
    lv_obj_add_event_cb(sw, event_handler, LV_EVENT_ALL, NULL);

    sw = lv_switch_create(lv_screen_active());
    lv_obj_set_size(sw, 30, 60);
    lv_switch_set_orientation(sw, LV_SWITCH_ORIENTATION_VERTICAL);
    lv_obj_add_state(sw, LV_STATE_CHECKED);
    lv_obj_add_event_cb(sw, event_handler, LV_EVENT_ALL, NULL);
}

#endif
```

6.30.7 API

lv_switch.h

lv_switch_private.h

6.31 Table (lv_table)

6.31.1 Panoramica

Tables are built from rows, columns, and cells containing text.

The Table Widget is very lightweight because only the text strings are stored. No real Widgets are created for cells --- they are just drawn on the fly.

The Table is added to the default group (if one is set). Table is an editable Widget, allowing selection of a cell with encoder and keyboard navigation as well.

6.31.2 Parts and Styles

- `LV_PART_MAIN` The background of the Table; uses the *typical background style properties*.
- `LV_PART_ITEMS` The cells of the Table also use the *typical background style properties* as well as text style properties.

6.31.3 Utilizzo

Set cell value

Cells can store only text so numbers need to be converted to text before displaying them in a Table.

`lv_table_set_cell_value(table, row, col, "Content")`. The text is saved by the Table so the buffer containing the string can be a local variable.

Line breaks can be used in the text like `"Value\n60.3"`.

New rows and columns are automatically added as required.

Rows and Columns

To explicitly set number of rows and columns use `lv_table_set_row_count(table, row_cnt)` and `lv_table_set_column_count(table, col_cnt)`.

Width and Height

Column width can be set with `lv_table_set_column_width(table, col_id, width)`. The overall width of the Table Widget will be set to the sum of all column widths.

Height is calculated automatically from the cell styles (font, padding etc) and the number of rows.

Merge cells

Cells can be merged horizontally with `lv_table_set_cell_ctrl(table, row, col, LV_TABLE_CELL_CTRL_MERGE_RIGHT)`. To merge more adjacent cells, call this function for each cell.

Scrolling

If a Table's width or height is set to `LV_SIZE_CONTENT` that size will be used to show the whole Table in the respective direction. E.g. `lv_obj_set_size(table, LV_SIZE_CONTENT, LV_SIZE_CONTENT)` automatically sets the Table size to show all columns and rows.

If the width or height is set to a smaller number than its "intrinsic" size then the Table becomes scrollable.

6.31.4 Eventi

- *LV_EVENT_VALUE_CHANGED* Sent when a new cell is selected with keys.

> **Further Reading**
>
> Learn more about *Base-Widget Events* emitted by all Widgets.
>
> Learn more about *Eventi*.

6.31.5 Keys

The following *Keys* are processed by Table Widgets:

- `LV_KEY_RIGHT/LEFT/UP/DOWN/` Select a cell.

Note that, as usual, the state of *LV_KEY_ENTER* is translated to `LV_EVENT_PRESSED/PRESSING/RELEASED` etc.

lv_table_get_selected_cell`(table, &row, &col)` can be used to get the currently selected cell. Row and column will be set to *LV_TABLE_CELL_NONE* if no cell is selected.

> **Further Reading**
>
> Learn more about *Keys*.

6.31.6 Esempio

Tabella semplice

```
#include "../../lv_examples.h"
#if LV_USE_TABLE && LV_BUILD_EXAMPLES

static void draw_event_cb(lv_event_t * e)
{
    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);
    /*If the cells are drawn...*/
    if(base_dsc->part == LV_PART_ITEMS) {
        uint32_t row = base_dsc->id1;
        uint32_t col = base_dsc->id2;

        /*Make the texts in the first cell center aligned*/
        if(row == 0) {
            lv_draw_label_dsc_t * label_draw_dsc = lv_draw_task_get_label_
↪dsc(draw_task);
            if(label_draw_dsc) {
                label_draw_dsc->align = LV_TEXT_ALIGN_CENTER;
            }
            lv_draw_fill_dsc_t * fill_draw_dsc = lv_draw_task_get_fill_
↪dsc(draw_task);
            if(fill_draw_dsc) {
                fill_draw_dsc->color = lv_color_mix(lv_palette_main(LV_
↪PALETTE_BLUE), fill_draw_dsc->color, LV_OPA_20);
                fill_draw_dsc->opa = LV_OPA_COVER;
            }
        }
```

(continues on next page)

(continua dalla pagina precedente)

```
        /*In the first column align the texts to the right*/
        else if(col == 0) {
            lv_draw_label_dsc_t * label_draw_dsc = lv_draw_task_get_label_
↪dsc(draw_task);
            if(label_draw_dsc) {
                label_draw_dsc->align = LV_TEXT_ALIGN_RIGHT;
            }
        }

        /*Make every 2nd row grayish*/
        if((row != 0 && row % 2) == 0) {
            lv_draw_fill_dsc_t * fill_draw_dsc = lv_draw_task_get_fill_
↪dsc(draw_task);
            if(fill_draw_dsc) {
                fill_draw_dsc->color = lv_color_mix(lv_palette_main(LV_
↪PALETTE_GREY), fill_draw_dsc->color, LV_OPA_10);
                fill_draw_dsc->opa = LV_OPA_COVER;
            }
        }
    }
}

void lv_example_table_1(void)
{
    lv_obj_t * table = lv_table_create(lv_screen_active());

    /*Fill the first column*/
    lv_table_set_cell_value(table, 0, 0, "Name");
    lv_table_set_cell_value(table, 1, 0, "Apple");
    lv_table_set_cell_value(table, 2, 0, "Banana");
    lv_table_set_cell_value(table, 3, 0, "Lemon");
    lv_table_set_cell_value(table, 4, 0, "Grape");
    lv_table_set_cell_value(table, 5, 0, "Melon");
    lv_table_set_cell_value(table, 6, 0, "Peach");
    lv_table_set_cell_value(table, 7, 0, "Nuts");

    /*Fill the second column*/
    lv_table_set_cell_value(table, 0, 1, "Price");
    lv_table_set_cell_value(table, 1, 1, "$7");
    lv_table_set_cell_value(table, 2, 1, "$4");
    lv_table_set_cell_value(table, 3, 1, "$6");
    lv_table_set_cell_value(table, 4, 1, "$2");
    lv_table_set_cell_value(table, 5, 1, "$5");
    lv_table_set_cell_value(table, 6, 1, "$1");
    lv_table_set_cell_value(table, 7, 1, "$9");

    /*Set a smaller height to the table. It'll make it scrollable*/
    lv_obj_set_height(table, 200);
    lv_obj_center(table);

    /*Add an event callback to to apply some custom drawing*/
    lv_obj_add_event_cb(table, draw_event_cb, LV_EVENT_DRAW_TASK_ADDED,␣
↪NULL);
    lv_obj_add_flag(table, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);
}
```

(continues on next page)

(continua dalla pagina precedente)

```
#endif
```

Elenco leggero da tabella

```
#include "../../lv_examples.h"
#if LV_USE_TABLE && LV_BUILD_EXAMPLES

#define ITEM_CNT 200

static void draw_event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);

    lv_draw_task_t * draw_task = lv_event_get_draw_task(e);
    lv_draw_dsc_base_t * base_dsc = (lv_draw_dsc_base_t *)lv_draw_task_get_
↪draw_dsc(draw_task);
    /*If the cells are drawn...*/
    if(base_dsc->part == LV_PART_ITEMS && lv_draw_task_get_type(draw_task)␣
↪== LV_DRAW_TASK_TYPE_FILL) {
        /*Draw the background*/
        bool chk = lv_table_has_cell_ctrl(obj, base_dsc->id1, 0, LV_TABLE_
↪CELL_CTRL_CUSTOM_1);
        lv_draw_rect_dsc_t rect_dsc;
        lv_draw_rect_dsc_init(&rect_dsc);
        rect_dsc.bg_color = chk ? lv_theme_get_color_primary(obj) : lv_
↪palette_lighten(LV_PALETTE_GREY, 2);
        rect_dsc.radius = LV_RADIUS_CIRCLE;

        lv_area_t sw_area;
        sw_area.x1 = 0;
        sw_area.x2 = 40;
        sw_area.y1 = 0;
        sw_area.y2 = 24;
        lv_area_t draw_task_area;
        lv_draw_task_get_area(draw_task, &draw_task_area);
        lv_area_align(&draw_task_area, &sw_area, LV_ALIGN_RIGHT_MID, -15,␣
↪0);
        lv_draw_rect(base_dsc->layer, &rect_dsc, &sw_area);

        /*Draw the knob*/
        rect_dsc.bg_color = lv_color_white();
        lv_area_t knob_area;
        knob_area.x1 = 0;
        knob_area.x2 = 18;
        knob_area.y1 = 0;
        knob_area.y2 = 18;
        if(chk) {
            lv_area_align(&sw_area, &knob_area, LV_ALIGN_RIGHT_MID, -3, 0);
        }
        else {
            lv_area_align(&sw_area, &knob_area, LV_ALIGN_LEFT_MID, 3, 0);
        }
        lv_draw_rect(base_dsc->layer, &rect_dsc, &knob_area);
    }
}
```

(continues on next page)

(continua dalla pagina precedente)

```
static void change_event_cb(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    uint32_t col;
    uint32_t row;
    lv_table_get_selected_cell(obj, &row, &col);
    bool chk = lv_table_has_cell_ctrl(obj, row, 0, LV_TABLE_CELL_CTRL_
↪CUSTOM_1);
    if(chk) lv_table_clear_cell_ctrl(obj, row, 0, LV_TABLE_CELL_CTRL_
↪CUSTOM_1);
    else lv_table_set_cell_ctrl(obj, row, 0, LV_TABLE_CELL_CTRL_CUSTOM_1);
}

/**
 * A very light-weighted list created from table
 */
void lv_example_table_2(void)
{
    /*Measure memory usage*/
    lv_mem_monitor_t mon1;
    lv_mem_monitor(&mon1);

    uint32_t t = lv_tick_get();

    lv_obj_t * table = lv_table_create(lv_screen_active());

    /*Set a smaller height to the table. It'll make it scrollable*/
    lv_obj_set_size(table, LV_SIZE_CONTENT, 200);

    lv_table_set_column_width(table, 0, 150);
    lv_table_set_row_count(table, ITEM_CNT); /*Not required but avoids a␣
↪lot of memory reallocation lv_table_set_set_value*/
    lv_table_set_column_count(table, 1);

    /*Don't make the cell pressed, we will draw something different in the␣
↪event*/
    lv_obj_remove_style(table, NULL, LV_PART_ITEMS | LV_STATE_PRESSED);

    uint32_t i;
    for(i = 0; i < ITEM_CNT; i++) {
        lv_table_set_cell_value_fmt(table, i, 0, "Item %" LV_PRIu32, i +␣
↪1);
    }

    lv_obj_align(table, LV_ALIGN_CENTER, 0, -20);

    /*Add an event callback to to apply some custom drawing*/
    lv_obj_add_event_cb(table, draw_event_cb, LV_EVENT_DRAW_TASK_ADDED,␣
↪NULL);
    lv_obj_add_event_cb(table, change_event_cb, LV_EVENT_VALUE_CHANGED,␣
↪NULL);
    lv_obj_add_flag(table, LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS);

    lv_mem_monitor_t mon2;
    lv_mem_monitor(&mon2);
```

(continues on next page)

(continua dalla pagina precedente)

```
    size_t mem_used = mon1.free_size - mon2.free_size;

    uint32_t elaps = lv_tick_elaps(t);

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text_fmt(label, "%" LV_PRIu32" items were created in %"␣
↪LV_PRIu32" ms\n"
                          "using %zu bytes of memory",
                          (uint32_t)ITEM_CNT, elaps, mem_used);

    lv_obj_align(label, LV_ALIGN_BOTTOM_MID, 0, -10);

}

#endif
```

MicroPython

No examples yet.

6.31.7 API

lv_api_map_v8.h

lv_table.h

lv_table_private.h

6.32 Tab View (lv_tabview)

6.32.1 Panoramica

The Tab View Widget can be used to organize content in tabs. The Tab View is built from other Widgets:

- Main container: *Base Widget (lv_obj)*
- Tab buttons: an *Base Widget (lv_obj)* with *Button (lv_button)*
- Container for the tabs: *Base Widget (lv_obj)*
- Content of the tabs: *Base Widget (lv_obj)*

The tab buttons can be positioned on the top, bottom, left, and right side of the Tab View.

A new tab can be selected either by clicking on a tab button or by sliding horizontally on the content.

6.32.2 Parts and Styles

There are no special parts on the Tab View but the *Base Widget (lv_obj)* and *Button (lv_button)* Widgets are used to create the Tab View.

6.32.3 Utilizzo

Creating a Tab View

`lv_tabview_create(parent)` creates a new empty Tab View.

Adding tabs

New tabs can be added with `lv_tabview_add_tab(tabview, "Tab name")`. This will return a pointer to a *Base Widget (lv_obj)* where the tab's content can be created.

Renaming tabs

A tab can be renamed with `lv_tabview_rename_tab(tabview, tab_id, "New Name")`.

Navigating to a new tab

To select a new tab you can:

- Click on its tab button
- Slide horizontally
- Use `lv_tabview_set_active(tabview, tab_id, LV_ANIM_ON / OFF)` function

Setting tab bar position

Using the `lv_tabview_set_tab_bar_position(tabview, LV_DIR_LEFT / RIGHT / TOP / BOTTOM)` the tab bar can be moved to any side.

Setting tab bar size

The size of the tab bar can be adjusted by `lv_tabview_set_tab_bar_size(tabview, size)` When tabs are on the top or bottom, this means the height of the tab bar, and when they are on the sides, it means the width.

Accessing the parts

- `lv_tabview_get_content(tabview)` returns a pointer to the container for tabs content (a *Base Widget (lv_obj)*)
- `lv_tabview_get_tab_bar(tabview)` returns a pointer to the container for tab buttons (a *Base Widget (lv_obj)*)

6.32.4 Eventi

- `LV_EVENT_VALUE_CHANGED` Sent when a new tab is selected by sliding or clicking the tab button. `lv_tabview_get_tab_active(tabview)` returns the zero based index of the current tab.

> **Further Reading**
>
> Learn more about *Base-Widget Events* emitted by all Widgets.
>
> Learn more about *Eventi*.

6.32.5 Keys

Keys have effect only on the tab buttons. Programmatically add these buttons to a group if required.

> **Further Reading**
>
> Learn more about *Keys*.

6.32.6 Esempio

Tabview Semplice

```
#include "../../lv_examples.h"
#if LV_USE_TABVIEW && LV_BUILD_EXAMPLES

void lv_example_tabview_1(void)
{
    /*Create a Tab view object*/
    lv_obj_t * tabview;
    tabview = lv_tabview_create(lv_screen_active());

    /*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);
    lv_label_set_text(label, "This the first tab\n\n"
                      "If the content\n"
                      "of a tab\n"
                      "becomes too\n"
                      "longer\n"
                      "than the\n"
                      "container\n"
                      "then it\n"
                      "automatically\n"
                      "becomes\n"
                      "scrollable.\n"
                      "\n"
                      "\n"
                      "\n"
                      "Can you see it?");

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

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

    lv_obj_scroll_to_view_recursive(label, LV_ANIM_ON);

}
#endif
```

Tab a sinistra, con stile e nessuno scorrimento

```
#include "../../lv_examples.h"
#if LV_USE_TABVIEW && LV_BUILD_EXAMPLES

/*A vertical tab view with disabled scrolling and some styling*/
void lv_example_tabview_2(void)
{
    /*Create a Tab view object*/
    lv_obj_t * tabview;
```

(continues on next page)

(continua dalla pagina precedente)

```
    uint32_t tab_count = 0;
    uint32_t i = 0;

    tabview = lv_tabview_create(lv_screen_active());
    lv_tabview_set_tab_bar_position(tabview, LV_DIR_LEFT);
    lv_tabview_set_tab_bar_size(tabview, 80);

    lv_obj_set_style_bg_color(tabview, lv_palette_lighten(LV_PALETTE_RED,␣
↪2), 0);

    lv_obj_t * tab_buttons = lv_tabview_get_tab_bar(tabview);
    lv_obj_set_style_bg_color(tab_buttons, lv_palette_darken(LV_PALETTE_
↪GREY, 3), 0);
    lv_obj_set_style_text_color(tab_buttons, lv_palette_lighten(LV_PALETTE_
↪GREY, 5), 0);

    /*Add 5 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");
    lv_obj_t * tab4 = lv_tabview_add_tab(tabview, "Tab 4");
    lv_obj_t * tab5 = lv_tabview_add_tab(tabview, "Tab 5");

    tab_count = lv_tabview_get_tab_count(tabview);
    for(i = 0; i < tab_count; i++) {
        lv_obj_t * button = lv_obj_get_child(tab_buttons, i);
        lv_obj_set_style_border_side(button, LV_BORDER_SIDE_RIGHT, LV_PART_
↪MAIN | LV_STATE_CHECKED);
    }
    lv_obj_set_style_bg_color(tab2, lv_palette_lighten(LV_PALETTE_AMBER,␣
↪3), 0);
    lv_obj_set_style_bg_opa(tab2, LV_OPA_COVER, 0);

    /*Add content to the tabs*/
    lv_obj_t * label = lv_label_create(tab1);
    lv_label_set_text(label, "First tab");

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

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

    label = lv_label_create(tab4);
    lv_label_set_text(label, "Fourth tab");

    label = lv_label_create(tab5);
    lv_label_set_text(label, "Fifth tab");

    lv_obj_remove_flag(lv_tabview_get_content(tabview), LV_OBJ_FLAG_
↪SCROLLABLE);
}
#endif
```

6.32.7 API

lv_api_map_v8.h

lv_tabview.h

lv_tabview_private.h

6.33 Text Area (lv_textarea)

6.33.1 Panoramica

The Text Area is a *Base Widget (lv_obj)* with a *Label (lv_label)* and a cursor on it. Text or characters can be added to it. Long lines are wrapped and when the text becomes long enough, the Text Area can be scrolled.

One-line mode and password modes are supported.

6.33.2 Parts and Styles

- `LV_PART_MAIN` The background of the Text Area; uses the *typical background style properties* and the text related style properties including `text_align` to align the text to the left, right or center.
- `LV_PART_SCROLLBAR` The scrollbar that is shown when the text is longer than its height.
- `LV_PART_SELECTED` Determines the style of the *selected text*. Only `text_color` and `bg_color` style properties can be used. `bg_color` should be set directly on the label of the Text Area.
- `LV_PART_CURSOR` Marks the position where the characters are inserted. The cursor's area is always the bounding box of the current character. A block cursor can be created by adding a background color and background opacity to `LV_PART_CURSOR`'s style. To create a "bar" cursor leave the cursor transparent and set a left border. The `anim_time` style property sets the cursor's blink time.
- `LV_PART_TEXTAREA_PLACEHOLDER` Unique to Text Area; allows styling the *placeholder text*.

6.33.3 Utilizzo

Adding text

You can insert text or characters to the current cursor's position with:

- `lv_textarea_add_char(textarea, 'c')`
- `lv_textarea_add_text(textarea, "insert this text")`

To add wide characters like `'á'`, `'ß'` or CJK characters, use `lv_textarea_add_text(textarea, "á")`.

`lv_textarea_set_text(textarea, "New text")` replaces all existing text with "New text".

Placeholder text

Placeholder text is text that is displayed when the Text Area is empty. This can be a handy way to provide the end user with a hint about what to type there.

Specify placeholder text using `lv_textarea_set_placeholder_text(textarea, "Placeholder text")`.

Delete character

To delete the character to the left of the current cursor position, use `lv_textarea_delete_char(textarea)`.

To delete to the right, use `lv_textarea_delete_char_forward(textarea)`

Moving the cursor

The cursor position can be modified programmatically using `lv_textarea_set_cursor_pos(textarea, cursor_pos)` where `cursor_pos` is the zero-based index of the character the cursor should be placed in front of. `LV_TEXTAREA_CURSOR_LAST` can be passed to mean "after the last character"

You can move the cursor one character-position (or line) at a time with

- `lv_textarea_cursor_right(textarea)`
- `lv_textarea_cursor_left(textarea)`
- `lv_textarea_cursor_up(textarea)`
- `lv_textarea_cursor_down(textarea)`

If `lv_textarea_set_cursor_click_pos(textarea, true)` is applied, the cursor will jump to the position where the Text Area was clicked.

Hiding the cursor

The cursor is normally always visible. It can be a good idea to style it to be visible only in `LV_STATE_FOCUSED` state. See *Stili* for more information about how to do this.

One-line mode

The Text Area can be configured to keep all text on a single line with `lv_textarea_set_one_line(textarea, true)`. In this mode:

- the height is set automatically to show only one line,
- line break characters are ignored, and
- word wrap is disabled.

Password mode

The Text Area supports password mode which can be enabled with `lv_textarea_set_password_mode(textarea, true)`.

By default, if the • (Bullet, U+2022) character exists in the font, the entered characters are converted to it after a configurable delay after each new character is entered. If • does not exist in the font, * will be used. You can override the default "masking" character with `lv_textarea_set_password_bullet(textarea, str)` where `str` is a NUL-terminated C string. Esempio:

```
lv_textarea_set_password_bullet(textarea, "x");
```

In password mode `lv_textarea_get_text(textarea)` returns the actual text entered, not the bullet characters.

The visibility time can be adjusted with `LV_TEXTAREA_DEF_PWD_SHOW_TIME` in `lv_conf.h`.

Accepted characters

You can set a list of accepted characters with `lv_textarea_set_accepted_chars(textarea, list)` where `list` is a pointer to a NUL-terminated string, or NULL to accept all characters. Characters entered not in this list will be ignored.

```
lv_textarea_set_accepted_chars(textarea, "0123456789.+-");
```

Max text length

The maximum number of characters can be limited using `lv_textarea_set_max_length(textarea, max_char_num)`.

Very long text

If the text in the Text Area is very long (e.g. > 20k characters), scrolling and drawing might be slow. However, by setting *LV_LABEL_LONG_TXT_HINT* in `lv_conf.h` to a non-zero value, the performance with long text is significantly improved. It does this by saving some additional information about the current vertical position of the text shown. With this mode configured, scrolling and drawing is as fast as with "normal" short text. The cost is 12 extra bytes per label in RAM.

This value is set to `1` by default. If you do not use long text, you can save 12 bytes per label by setting it to `0`.

Selecting text

If *LV_LABEL_TEXT_SELECTION* is set to a non-zero value in `lv_conf.h`, some additional functionality (and 8 bytes per label) are added to Label Widgets and Text Area Widgets, and text-selection functionality is automated in Text Area Widgets. (If you do not use selected text in your application, you can save 8 bytes per label in RAM by setting that macro to equate to `0`.)

Any part of the text can be selected if enabled with *lv_textarea_set_text_selection*`(textarea, true)`. This works much like when you select text on your PC by clicking and dragging with your mouse or other pointer input device. If you pass `false` to this function to disable text selection, any text selected time of the call will be de-selected.

Shift+click and keyboard-only text selection with Shift+Arrow keys is expected to be added in the future.

If you need to programmatically deal with selected text, in addition to the *lv_textarea_set_text_selection*`(textarea, enable)` function, the following is your tool set for doing so. (`ta_label` is a pointer to the Text Area's Label retrieved with `ta_label = lv_textarea_get_label(textarea);`.)

- *lv_textarea_get_text_selection*`(textarea)` tells whether text selection is enabled.
- *lv_textarea_text_is_selected*`(textarea)` tells whether any text is currently selected.
- *lv_textarea_clear_selection*`(textarea)` clears current text selection.
- *lv_label_set_text_selection_start*`(ta_label, index)` where `index` is the zero-based index of the first character of the selected text. Pass *LV_DRAW_LABEL_NO_TXT_SEL* to specify no text selected.
- *lv_label_set_text_selection_end*`(ta_label, index)` where `index` is the zero-based index of the character just after the selected text. Pass *LV_DRAW_LABEL_NO_TXT_SEL* to specify no text selected.
- *lv_label_get_text_selection_start*`(ta_label)` zero-based index of the first character of the selected text. *LV_DRAW_LABEL_NO_TXT_SEL* indicates no text selected.
- *lv_label_get_text_selection_end*`(ta_label)` zero-based index of the character just after the selected text. *LV_DRAW_LABEL_NO_TXT_SEL* indicates no text selected.

Normally you won't need these since Text Area automates the text selection process, but if you do need to change the selection programmatically, the above are your tools to do so.

6.33.4 Eventi

- *LV_EVENT_INSERT* Sent right before a character or text is inserted. The event parameter is the text about to be inserted. *lv_textarea_set_insert_replace*`(textarea, "New text")` can be called from within that event to replace the text to be inserted. The contents of the buffer passed must be survive long enough for the call to `lv_timer_handler()` that is driving the event to return (after which the Text Area's label will have copied the text). So it should not be a local buffer (created on the stack) where its contents will be destroyed before that happens. Passing `""` means "do not insert anything".
- *LV_EVENT_VALUE_CHANGED* Sent when the content of the Text Area has changed.
- *LV_EVENT_READY* Sent when *LV_KEY_ENTER* is pressed (or sent) to a one-line Text Area.

Further Reading

Learn more about *Base-Widget Events* emitted by all Widgets.

Learn more about *Eventi*.

6.33.5 Keys

- `LV_KEY_UP/DOWN/LEFT/RIGHT` Move the cursor
- `Any character` Add the character to the current cursor position

Further Reading

Learn more about *Keys*.

6.33.6 Esempio

Area di testo semplice

```
#include "../../lv_examples.h"
#if LV_USE_TEXTAREA && LV_BUILD_EXAMPLES

static void textarea_event_handler(lv_event_t * e)
{
    lv_obj_t * ta = lv_event_get_target_obj(e);
    LV_UNUSED(ta);
    LV_LOG_USER("Enter was pressed. The current text is: %s", lv_textarea_
↪get_text(ta));
}

static void btnm_event_handler(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    lv_obj_t * ta = (lv_obj_t *)lv_event_get_user_data(e);
    const char * txt = lv_buttonmatrix_get_button_text(obj, lv_
↪buttonmatrix_get_selected_button(obj));

    if(lv_strcmp(txt, LV_SYMBOL_BACKSPACE) == 0) lv_textarea_delete_
↪char(ta);
    else if(lv_strcmp(txt, LV_SYMBOL_NEW_LINE) == 0) lv_obj_send_event(ta,␣
↪LV_EVENT_READY, NULL);
    else lv_textarea_add_text(ta, txt);

}

void lv_example_textarea_1(void)
{
    lv_obj_t * ta = lv_textarea_create(lv_screen_active());
    lv_textarea_set_one_line(ta, true);
    lv_obj_align(ta, LV_ALIGN_TOP_MID, 0, 10);
    lv_obj_add_event_cb(ta, textarea_event_handler, LV_EVENT_READY, ta);
    lv_obj_add_state(ta, LV_STATE_FOCUSED); /*To be sure the cursor is␣
↪visible*/

    static const char * btnm_map[] = {"1", "2", "3", "\n",
```

(continues on next page)

(continua dalla pagina precedente)

```
                                        "4", "5", "6", "\n",
                                        "7", "8", "9", "\n",
                                        LV_SYMBOL_BACKSPACE, "0", LV_SYMBOL_
↪NEW_LINE, ""
                                       };

    lv_obj_t * btnm = lv_buttonmatrix_create(lv_screen_active());
    lv_obj_set_size(btnm, 200, 150);
    lv_obj_align(btnm, LV_ALIGN_BOTTOM_MID, 0, -10);
    lv_obj_add_event_cb(btnm, btnm_event_handler, LV_EVENT_VALUE_CHANGED,␣
↪ta);
    lv_obj_remove_flag(btnm, LV_OBJ_FLAG_CLICK_FOCUSABLE); /*To keep the␣
↪text area focused on button clicks*/
    lv_buttonmatrix_set_map(btnm, btnm_map);
}

#endif
```

Area di testo con campo password

```
#include "../../lv_examples.h"
#if LV_USE_TEXTAREA && LV_USE_KEYBOARD && LV_BUILD_EXAMPLES

static void ta_event_cb(lv_event_t * e);

static lv_obj_t * kb;

void lv_example_textarea_2(void)
{
    /*Create the password box*/
    lv_obj_t * pwd_ta = lv_textarea_create(lv_screen_active());
    lv_textarea_set_text(pwd_ta, "");
    lv_textarea_set_password_mode(pwd_ta, true);
    lv_textarea_set_one_line(pwd_ta, true);
    lv_obj_set_width(pwd_ta, lv_pct(40));
    lv_obj_set_pos(pwd_ta, 5, 20);
    lv_obj_add_event_cb(pwd_ta, ta_event_cb, LV_EVENT_ALL, NULL);

    /*Create a label and position it above the text box*/
    lv_obj_t * pwd_label = lv_label_create(lv_screen_active());
    lv_label_set_text(pwd_label, "Password:");
    lv_obj_align_to(pwd_label, pwd_ta, LV_ALIGN_OUT_TOP_LEFT, 0, 0);

    /*Create the one-line mode text area*/
    lv_obj_t * text_ta = lv_textarea_create(lv_screen_active());
    lv_textarea_set_one_line(text_ta, true);
    lv_textarea_set_password_mode(text_ta, false);
    lv_obj_set_width(text_ta, lv_pct(40));
    lv_obj_add_event_cb(text_ta, ta_event_cb, LV_EVENT_ALL, NULL);
    lv_obj_align(text_ta, LV_ALIGN_TOP_RIGHT, -5, 20);

    /*Create a label and position it above the text box*/
    lv_obj_t * oneline_label = lv_label_create(lv_screen_active());
    lv_label_set_text(oneline_label, "Text:");
    lv_obj_align_to(oneline_label, text_ta, LV_ALIGN_OUT_TOP_LEFT, 0, 0);
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Create a keyboard*/
    kb = lv_keyboard_create(lv_screen_active());
    lv_obj_set_size(kb,  LV_HOR_RES, LV_VER_RES / 2);

    lv_keyboard_set_textarea(kb, pwd_ta); /*Focus it on one of the text␣
↪areas to start*/

    /*The keyboard will show Arabic characters if they are enabled */
#if LV_USE_ARABIC_PERSIAN_CHARS && LV_FONT_DEJAVU_16_PERSIAN_HEBREW
    lv_obj_set_style_text_font(kb, &lv_font_dejavu_16_persian_hebrew, 0);
    lv_obj_set_style_text_font(text_ta, &lv_font_dejavu_16_persian_hebrew,␣
↪0);
    lv_obj_set_style_text_font(pwd_ta, &lv_font_dejavu_16_persian_hebrew,␣
↪0);
#endif

}

static void ta_event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * ta = lv_event_get_target_obj(e);
    if(code == LV_EVENT_CLICKED || code == LV_EVENT_FOCUSED) {
        /*Focus on the clicked text area*/
        if(kb != NULL) lv_keyboard_set_textarea(kb, ta);
    }

    else if(code == LV_EVENT_READY) {
        LV_LOG_USER("Ready, current text: %s", lv_textarea_get_text(ta));
    }
}

#endif
```

Formattazione automatica del testo

```
#include "../../lv_examples.h"
#if LV_USE_TEXTAREA && LV_USE_KEYBOARD && LV_BUILD_EXAMPLES

static void ta_event_cb(lv_event_t * e);

static lv_obj_t * kb;

/**
 * Automatically format text like a clock. E.g. "12:34"
 * Add the ':' automatically.
 */
void lv_example_textarea_3(void)
{
    /*Create the text area*/
    lv_obj_t * ta = lv_textarea_create(lv_screen_active());
    lv_obj_add_event_cb(ta, ta_event_cb, LV_EVENT_VALUE_CHANGED, NULL);
    lv_textarea_set_accepted_chars(ta, "0123456789:");
    lv_textarea_set_max_length(ta, 5);
    lv_textarea_set_one_line(ta, true);
    lv_textarea_set_text(ta, "");
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Create a keyboard*/
    kb = lv_keyboard_create(lv_screen_active());
    lv_obj_set_size(kb,  LV_HOR_RES, LV_VER_RES / 2);
    lv_keyboard_set_mode(kb, LV_KEYBOARD_MODE_NUMBER);
    lv_keyboard_set_textarea(kb, ta);
}

static void ta_event_cb(lv_event_t * e)
{
    lv_obj_t * ta = lv_event_get_target_obj(e);
    const char * txt = lv_textarea_get_text(ta);
    if(txt[0] >= '0' && txt[0] <= '9' &&
       txt[1] >= '0' && txt[1] <= '9' &&
       txt[2] != ':') {
        lv_textarea_set_cursor_pos(ta, 2);
        lv_textarea_add_char(ta, ':');
    }
}

#endif
```

Stile del cursore dell'area di testo

```
#include "../../lv_examples.h"
#if LV_USE_TEXTAREA && LV_USE_KEYBOARD && LV_BUILD_EXAMPLES

static void create_styled_textarea_cursor(const char * txt, lv_coord_t y_
↪ofs, lv_style_t * cursor_style)
{
    lv_obj_t * ta = lv_textarea_create(lv_screen_active());
    lv_textarea_set_text(ta, txt);
    lv_obj_set_width(ta, 280);
    lv_obj_align(ta, LV_ALIGN_TOP_MID, 0, y_ofs);
    lv_textarea_set_one_line(ta, true);
    lv_obj_add_state(ta, LV_STATE_FOCUSED);
    lv_obj_add_style(ta, cursor_style, LV_PART_CURSOR | LV_STATE_FOCUSED);
    lv_textarea_set_cursor_pos(ta, 0);
}

void lv_example_textarea_4(void)
{
    static lv_style_t style_simple, style_block, style_underline;

    /* Thin left bar cursor (simple) */
    lv_style_init(&style_simple);
    lv_style_set_border_color(&style_simple, lv_palette_main(LV_PALETTE_
↪RED));

    /* Underline cursor */
    lv_style_init(&style_underline);
    lv_style_set_bg_opa(&style_underline, LV_OPA_TRANSP);
    lv_style_set_border_color(&style_underline, lv_palette_main(LV_PALETTE_
↪BLUE));
    lv_style_set_border_side(&style_underline, LV_BORDER_SIDE_BOTTOM);
    lv_style_set_pad_hor(&style_underline, 1); /* set width of cursor␣
```

(continues on next page)

(continua dalla pagina precedente)

```
↪using pad */
    lv_style_set_border_width(&style_underline, 3); /* set thickness of␣
↪underline cursor */

    /* Full block cursor with many styles */
    lv_style_init(&style_block);
    lv_style_set_bg_opa(&style_block, LV_OPA_COVER);
    lv_style_set_bg_color(&style_block, lv_palette_main(LV_PALETTE_
↪ORANGE));
    lv_style_set_bg_grad_color(&style_block, lv_palette_main(LV_PALETTE_
↪YELLOW));
    lv_style_set_bg_grad_dir(&style_block, LV_GRAD_DIR_VER);
    lv_style_set_border_color(&style_block, lv_palette_main(LV_PALETTE_
↪RED));
    lv_style_set_border_side(&style_block, LV_BORDER_SIDE_FULL);
    lv_style_set_border_width(&style_block, 1);
    lv_style_set_radius(&style_block, 4);
    lv_style_set_text_color(&style_block, lv_color_white());
    lv_style_set_pad_all(&style_block, 1); /* set width of cursor using␣
↪pad */

    /* Create 3 independent textareas, each with a unique styled cursor */
    create_styled_textarea_cursor("This is a simple red cursor", 10,  &
↪style_simple);
    create_styled_textarea_cursor("This is an underline blue cursor", 110,␣
↪&style_underline);
    create_styled_textarea_cursor("This is a complex block cursor", 60, &
↪style_block);
}

#endif
```

6.33.7 API

lv_obj_property_names.h

lv_textarea.h

lv_textarea_private.h

6.34 Tile View (lv_tileview)

6.34.1 Panoramica

The Tile View is a container Widget whose elements (called *tiles*) can be arranged in grid form. A user can navigate between the tiles by swiping. Any direction of swiping can be disabled on the tiles individually to not allow moving from one tile to another.

If the Tile View is screen sized, the user interface resembles what you may have seen on smartwatches.

6.34.2 Parts and Styles

The Tile View is built from a *Base Widget (lv_obj)* container and *Base Widget (lv_obj)* tiles.

The parts and styles work the same as for *Base Widget (lv_obj)*.

6.34.3 Utilizzo

Add a tile

`lv_tileview_add_tile(tileview, col_id, row_id, dir)` creates a new tile on the `col_id`-th column and `row_id`-th row. `dir` can be `LV_DIR_LEFT/RIGHT/TOP/BOTTOM/HOR/VER/ALL` or OR-ed values to enable moving to the adjacent tiles in the given direction by swiping.

The returned value is a pointer to the tile (a *Base Widget (lv_obj)*) on which the content of the tab can be created.

Change tile

The Tile View can scroll to a specified tile with `lv_tileview_set_tile(tileview, tile, LV_ANIM_ON / OFF)` or `lv_tileview_set_tile_by_index(tileview, col_id, row_id, LV_ANIM_ON / OFF)`.

6.34.4 Eventi

- `LV_EVENT_VALUE_CHANGED` Sent after a new tile is displayed by scrolling. `lv_tileview_get_tile_active(tileview)` can be used within the event to get a pointer to the newly-displayed tile.

> **Further Reading**
>
> Learn more about *Base-Widget Events* emitted by all Widgets.
>
> Learn more about *Eventi*.

6.34.5 Keys

No *Keys* are processed by Tileview Widgets.

> **Further Reading**
>
> Learn more about *Keys*.

6.34.6 Esempio

Tileview con contenuto

```
#include "../../lv_examples.h"
#if LV_USE_TILEVIEW && LV_BUILD_EXAMPLES

/**
 * Create a 2x2 tile view and allow scrolling only in an "L" shape.
 * Demonstrate scroll chaining with a long list that
 * scrolls the tile view when it can't be scrolled further.
 */
void lv_example_tileview_1(void)
{
    lv_obj_t * tv = lv_tileview_create(lv_screen_active());

    /*Tile1: just a label*/
    lv_obj_t * tile1 = lv_tileview_add_tile(tv, 0, 0, LV_DIR_BOTTOM);
    lv_obj_t * label = lv_label_create(tile1);
    lv_label_set_text(label, "Scroll down");
    lv_obj_center(label);
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Tile2: a button*/
    lv_obj_t * tile2 = lv_tileview_add_tile(tv, 0, 1, (lv_dir_t)(LV_DIR_
↪TOP | LV_DIR_RIGHT));

    lv_obj_t * btn = lv_button_create(tile2);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Scroll up or right");

    lv_obj_set_size(btn, LV_SIZE_CONTENT, LV_SIZE_CONTENT);
    lv_obj_center(btn);

    /*Tile3: a list*/
    lv_obj_t * tile3 = lv_tileview_add_tile(tv, 1, 1, LV_DIR_LEFT);
    lv_obj_t * list = lv_list_create(tile3);
    lv_obj_set_size(list, LV_PCT(100), LV_PCT(100));

    lv_list_add_button(list, NULL, "One");
    lv_list_add_button(list, NULL, "Two");
    lv_list_add_button(list, NULL, "Three");
    lv_list_add_button(list, NULL, "Four");
    lv_list_add_button(list, NULL, "Five");
    lv_list_add_button(list, NULL, "Six");
    lv_list_add_button(list, NULL, "Seven");
    lv_list_add_button(list, NULL, "Eight");
    lv_list_add_button(list, NULL, "Nine");
    lv_list_add_button(list, NULL, "Ten");

}

#endif
```

6.34.7 API

lv_api_map_v8.h

lv_tileview.h

lv_tileview_private.h

6.35 Window (lv_win)

6.35.1 Panoramica

The Window Widget is built from a header (like a title bar) with title and buttons and a content area.

6.35.2 Parts and Styles

The Window is built from other Widgets so you can check their documentation for details:

- Background: *Base Widget (lv_obj)*
- Header on the background: *Base Widget (lv_obj)*
- Title on the header: *Label (lv_label)*
- Buttons on the header: *Button (lv_button)*

- Content Area on the background: *Base Widget (lv_obj)*

6.35.3 Utilizzo

Create a Window

`lv_win_create(parent)` creates a Window that is initially composed of the following Widget structure:

- Background (a *Base Widget (lv_obj)*, the main window container), is set up to be a Flex-Flow container that flows its contained Widgets vertically (`LV_FLEX_FLOW_COLUMN`).
- Header (like a title bar) is initially empty, and is a Flex-Flow container set up to flow its contained Widgets horizontally (`LV_FLEX_FLOW_ROW`), left to right. The Header occupies the full width of the Background (its parent) and the top approximately 1/2 inch (according to `LV_DPI_DEF`).
- Content Area (a *Base Widget (lv_obj)*) occupies the full width of the Background (its parent) the remainder of the available Background area below the Header. It is *not itself* a Flex-Flow container, but you can make it so if you wish. See *Flex* for details.

Title and buttons

You can add Button and Label Widgets to the Header using these two functions. They will be placed in the Header in left-to-right order as they are added. These functions can be called in any order, any number of times.

- `lv_win_add_title(win, "The title")` adds a Label to the header, with long mode `LV_LABEL_LONG_DOT` so that if its text contents are wider than the area it has, the text will be truncated with an ellipsis ("...") placed at the end of the text. It is set to be FLEX GROW 1, so if it is the only Label in the header, it will occupy all available space after any Buttons are added. If more than one label is added, each label will share that space equally unless its FLEX GROW value is altered. See *Flex* for details about how this works. Because of this, any buttons added after the last Label added will be "stacked" on the far right of the Header.
- `lv_win_add_button(win, icon, button_width)` adds a Button with the specified width that occupies the full height of the Header (its parent). If `icon` is not NULL, an image is created, centered on the button, using `icon` as its image source. All valid image sources are supported, but a common source to use is one of the `LV_SYMBOL_...` macros, such as `LV_SYMBOL_CLOSE` to provide an "x" (close) symbol. You get back a pointer to the Button created so you can add an event callback with it and/or whatever else might be needed.

6.35.4 Getting the parts

`lv_win_get_header(win)` returns a pointer to the header, `lv_win_get_content(win)` returns a pointer to the content container to which the content of the window can be added.

6.35.5 Eventi

No special events are sent by Window Widgets, however events can be added to any Buttons added.

Further Reading

Learn more about *Base-Widget Events* emitted by all Widgets.

Learn more about *Eventi*.

6.35.6 Keys

No *Keys* are processed by Window Widgets.

Further Reading

Learn more about *Keys*.

6.35.7 Esempio

Window semplice

```
#include "../../lv_examples.h"
#if LV_USE_WIN && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    LV_UNUSED(obj);
    LV_LOG_USER("Button %d clicked", (int)lv_obj_get_index(obj));
}

void lv_example_win_1(void)
{
    lv_obj_t * win = lv_win_create(lv_screen_active());
    lv_obj_t * btn;
    btn = lv_win_add_button(win, LV_SYMBOL_LEFT, 40);
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);

    lv_win_add_title(win, "A title");

    btn = lv_win_add_button(win, LV_SYMBOL_RIGHT, 40);
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);

    btn = lv_win_add_button(win, LV_SYMBOL_CLOSE, 60);
    lv_obj_add_event_cb(btn, event_handler, LV_EVENT_CLICKED, NULL);

    lv_obj_t * cont = lv_win_get_content(win);  /*Content can be added␣
↪here*/
    lv_obj_t * label = lv_label_create(cont);
    lv_label_set_text(label, "This is\n"
                      "a pretty\n"
                      "long text\n"
                      "to see how\n"
                      "the window\n"
                      "becomes\n"
                      "scrollable.\n"
                      "\n"
                      "\n"
                      "Some more\n"
                      "text to be\n"
                      "sure it\n"
                      "overflows. :)");
}

#endif
```

6.35.8 API

lv_win.h

lv_win_private.h

lv_windows_context.h

lv_windows_display.h

lv_windows_input.h

6.36 New Widget

CAPITOLO 7

Main Modules

7.1 Display (lv_display)

7.1.1 Panoramica

What is a Display?

In LVGL, an *lv_display_t* (not to be confused with a *Screen*) is a data type that represents a single display panel --- the hardware that displays LVGL-rendered pixels on your device.

How Many Displays Can LVGL Use?

LVGL can use any number of displays. It is only limited by available RAM and MCU time.

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

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

If you set up LVGL to use more than one display, be aware that some functions use the *Display di Default* during their execution, such as creating *Screen*.

Attributes

Once created, a Display object remembers the characteristics of the display hardware it is representing, as well as other things relevant to its lifetime:

- Resolution (width and height in pixels)
- Color Depth (bits per pixel)
- Color Format (how colors in pixels are laid out)
- DPI (default is configured `LV_DPI_DEF` in `lv_conf.h`, but can be modified with `lv_display_set_dpi(disp, new_dpi)`).
- 4 *Screen Layers* automatically created with each display

- All *Screen* created in association with this display (and not yet deleted---only one is displayed at any given time)
- The *Draw Buffer(s)* assigned to it
- The *Flush Callback* function that moves pixels from *Draw Buffer(s)* to Display hardware
- What areas of the display have been updated (made "dirty") so rendering logic can compute what to render during a *display refresh*
- Optional custom pointer as *User Data*

User Data

With `lv_display_set_user_data(display1, p)` a custom pointer can be stored with `lv_display_t` object. This pointer can be used later, e.g. in *Display Events*. See code example for how to do this in *Display Events*.

API

lv_display.h

7.1.2 Setting Up Your Display(s)

During system initialization, you must do the following for each physical display panel you want LVGL to use:

- *create an lv_display_t* object for it,
- assign its *Draw Buffer(s)*, and
- assign a *Flush Callback* for it.

Creating a Display

To create a display for LVGL:

```
lv_display_t * display1 = lv_display_create(hor_res, ver_res)
```

You can create *multiple displays* with a different driver for each (see below).

When an `lv_display_t` object is created, with it are created 4 Screens set up to help you manage layering of displayed Widgets. See *Transparent Screens* and *Screen Layers* for more information.

Display di Default

Quando viene creato il primo oggetto *Display (lv_display)*, questo diventa il Display di Default. If other Display Objects are created (to service additional Display Panels), the Default Display remains the first one created.

To set another *Display (lv_display)* as the Default Display, call `lv_display_set_default()`.

See *How Many Displays Can LVGL Use?* for more information about using multiple displays.

For many `lv_display_...()` functions, passing NULL for the `disp` argument will cause the function to target the Default Display. Check the API documentation for the function you are calling to be sure.

Draw Buffer(s)

During system initialization, you must set drawing buffers for LVGL to use for each display. Do so by calling:

```
lv_display_set_buffers(display1, buf1, buf2, buf_size_in_bytes, render_
↪mode)
```

- `buf1` a buffer to which LVGL can render pixels
- `buf2` a second optional buffer (see below)
- `buf_size_in_bytes` size of buffer(s) in bytes

- `render_mode` is one of the following:
  - *LV_DISPLAY_RENDER_MODE_PARTIAL* Use the buffer(s) to render to the display using buffers smaller than the size of the display. Use of buffers at least 1/10 display size is recommended. In *Flush Callback* the rendered images need to be copied to the given area of the display. In this mode if a button is pressed only the button's area will be redrawn.
  - *LV_DISPLAY_RENDER_MODE_DIRECT* The buffer size(s) must match the size of the display. LVGL will render into the correct location of the buffer. Using this method the buffer(s) always contain the whole display image. If two buffers are used, the rendered areas are automatically copied to the other buffer after flushing. Due to this in *Flush Callback* typically only a frame buffer address needs to be changed. If a button is pressed only the button's area will be redrawn.
  - *LV_DISPLAY_RENDER_MODE_FULL* The buffer size(s) must match the size of the display. LVGL will always redraw the whole screen even if only 1 pixel has been changed. If two display-sized draw buffers are provided, LVGL's display handling works like "traditional" double buffering. This means the *Flush Callback* callback only has to update the address of the frame buffer to the `px_map` parameter.

Simple Example

```
/* Declare buffer for 1/10 screen size; BYTES_PER_PIXEL will be 2 for␣
↪RGB565. */
#define BYTES_PER_PIXEL (LV_COLOR_FORMAT_GET_SIZE(LV_COLOR_FORMAT_RGB565))
static uint8_t buf1[MY_DISP_HOR_RES * MY_DISP_VER_RES / 10 * BYTES_PER_
↪PIXEL];
/* Set display buffer for display `display1`. */
lv_display_set_buffers(display1, buf1, NULL, sizeof(buf1), LV_DISPLAY_
↪RENDER_MODE_PARTIAL);
```

One Buffer

If only one buffer is used, LVGL draws the content of the screen into that draw buffer and sends it to the display via the *Flush Callback*. LVGL then waits until *lv_display_flush_ready()* is called (that is, the content of the buffer has been sent to the display) before drawing something new into it.

Two Buffers

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

Three Buffers

Triple buffering enhances parallelism between rendering and data transfer compared to double buffering. When one buffer has completed rendering and another is actively undergoing DMA transfer, the third buffer enables immediate rendering of the next frame, eliminating CPU/GPU idle time caused by waiting for DMA completion. The third buffer is configured using the *lv_display_set_3rd_draw_buffer()* function.

Flush Callback

Draw buffer(s) are simple array(s) that LVGL uses to render the display's content. Once rendering is has been completed, the content of the draw buffer is sent to the display using a Flush Callback function.

An example looks like this:

```
void my_flush_cb(lv_display_t * display, const lv_area_t * area, uint8_t *␣
↪px_map)
{
    /* The most simple case (also the slowest) to send all rendered pixels␣
```

(continues on next page)

(continua dalla pagina precedente)

```
↪to the
     * screen one-by-one.  `put_px` is just an example.  It needs to be␣
↪implemented by you. */
    uint16_t * buf16 = (uint16_t *)px_map; /* Let's say it's a 16 bit␣
↪(RGB565) display */
    int32_t x, y;
    for(y = area->y1; y <= area->y2; y++) {
        for(x = area->x1; x <= area->x2; x++) {
            put_px(x, y, *buf16);
            buf16++;
        }
    }

    /* IMPORTANT!!!
     * Inform LVGL that flushing is complete so buffer can be modified␣
↪again. */
    lv_display_flush_ready(display);
}
```

During system initialization, tell LVGL you want that function to copy pixels from rendered pixel-buffers to a particular display by doing the following:

```
lv_display_set_flush_cb(display1, my_flush_cb)
```

Note that which display is targeted is passed to the function, so you can use the same function for multiple displays, or use different functions for multiple displays. It's up to you.

> **Nota**
>
> *lv_display_flush_ready*(display1) needs to be called when flushing is complete to inform LVGL that the buffer is available again to render new content into it.

LVGL might render the screen in multiple chunks and therefore call your Flush Callback multiple times. To see whether the current call is for the last chunk being rendered, use *lv_display_flush_is_last*(display1).

Flush-Wait Callback

By using *lv_display_flush_ready()* LVGL will normally spin in a loop while waiting for flushing.

However with the help of *lv_display_set_flush_wait_cb()* a custom wait callback be set for flushing. This callback can use a semaphore, mutex, or anything else to optimize waiting for the flush to be completed. The callback need not call *lv_display_flush_ready()* since the caller takes care of that (clearing the display's flushing flag) when your callback returns.

However, if a Flush-Wait Callback is not set, LVGL assumes that *lv_display_flush_ready()* is called after the flush has completed.

API

lv_display.h

lv_display_private.h

7.1.3 Screen Layers

When an `lv_display_t` object is created, 4 permanent *Screen* that facilitate layering are created and attached to it.

1. Bottom Layer (below Active Screen, transparent, not scroll-able, but click-able)
2. *Lo Screen Attivo*
3. Top Layer (above Active Screen, transparent and neither scroll-able nor click-able)
4. System Layer (above Top Layer, transparent and neither scroll-able nor click-able)

1, 3 and 4 are independent of the *Lo Screen Attivo* and they will be shown (if they contain anything that is visible) regardless of which screen is the *Lo Screen Attivo*.

> **Nota**
>
> For the bottom layer to be visible, the Active Screen's background has to be at least partially, if not fully, transparent.

You can get pointers to each of these screens on the *Display di Default* by using (respectively):

- `lv_screen_active()`,
- `lv_layer_top()`,
- `lv_layer_sys()`, and
- `lv_layer_bottom()`.

You can get pointers to each of these screens on a specified display by using (respectively):

- `lv_display_get_screen_active(disp)`,
- `lv_display_get_layer_top(disp)`,
- `lv_display_get_layer_sys(disp)`, and
- `lv_display_get_layer_bottom(disp)`.

To set a Screen you create to be the *Lo Screen Attivo*, call `lv_screen_load()` or `lv_screen_load_anim()`.

Top and System Layers

LVGL uses the Top Layer and System Layer to empower you to ensure that certain *All Widgets* are *always* on top of other layers.

You can add "pop-up windows" to the *Top Layer* freely. The Top Layer was meant to be used to create Widgets that are visible on all Screens shown on a Display. But, the *System Layer* is intended for system-level things (e.g. mouse cursor will be placed there with `lv_indev_set_cursor()`).

These layers work like any other Widget, meaning they have styles, and any kind of Widgets can be created in them.

> **Nota**
>
> While the Top Layer and System Layer are created by their owning *Display (lv_display)* as not scroll-able and not click-able, these behaviors can be overridden the same as any other Widget by using `lv_obj_set_scrollbar_mode(scr1, LV_SCROLLBAR_MODE_xxx)` and `lv_obj_add_flag(scr1, LV_OBJ_FLAG_CLICKABLE)` respectively.

If the `LV_OBJ_FLAG_CLICKABLE` flag is set on the Top Layer, then it will absorb all user clicks and acts as a modal Widget.

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

Bottom Layer

Similar to the Top- and System Layers, the Bottom Layer is also the full size of the Display, but it is located below the *Lo Screen Attivo*. It's visible only if the Active Screen's background opacity is < 255.

> **Further Reading**
>
> *Transparent Screens*.

API

lv_display.h

7.1.4 Color Format

The default color format of the display is set according to `LV_COLOR_DEPTH` (see `lv_conf.h`)

- `LV_COLOR_DEPTH 32`: XRGB8888 (4 bytes/pixel)
- `LV_COLOR_DEPTH 24`: RGB888 (3 bytes/pixel)
- `LV_COLOR_DEPTH 16`: RGB565 (2 bytes/pixel)
- `LV_COLOR_DEPTH 8`: L8 (1 bytes/pixel)
- `LV_COLOR_DEPTH 1`: I1 (1 bit/pixel) Only support for horizontal mapped buffers. See *Monochrome Displays* for more details:

The `color_format` can be changed with `lv_display_set_color_depth(display, LV_COLOR_FORMAT_...)`. Besides the default value `LV_COLOR_FORMAT_ARGB8888`, other formats can be used as well.

It's very important that draw buffer(s) should be large enough for the selected color format.

Swapping Endian-ness

In case of RGB565 color format it might be required to swap the 2 bytes because the SPI, I2C or 8 bit parallel port periphery sends them in the wrong order.

The ideal solution is to configure the hardware to handle the 16-bit data with a different byte order, however if this is not possible `lv_draw_sw_rgb565_swap(buf, buf_size_in_px)` can be called in the *Flush Callback* to swap the bytes.

If you wish you can also write your own function, or use assembly instructions for the fastest possible byte swapping.

Note that this is not about swapping the Red and Blue channel but converting

`RRRRR GGG | GGG BBBBB`

to

`GGG BBBBB | RRRRR GGG`.

Monochrome Displays

LVGL supports rendering directly in a 1-bit format for monochrome displays. To enable it, set `LV_COLOR_DEPTH 1` or use `lv_display_set_color_format(display, LV_COLOR_FORMAT_I1)`.

The `LV_COLOR_FORMAT_I1` format assumes that bytes are mapped to rows (i.e., the bits of a byte are written next to each other). The order of bits is MSB first, which means:

```
          MSB           LSB
bits       7 6 5 4 3 2 1 0
```

are represented on the display as:

```
pixels     0 1 2 3 4 5 6 7
          Left         Right
```

Ensure that the LCD controller is configured accordingly.

Internally, LVGL rounds the redrawn areas to byte boundaries. Therefore, updated areas will:

- start on an `Nx8` coordinate, and
- end on an `Nx8 - 1` coordinate.

When setting up the buffers for rendering (*`lv_display_set_buffers()`*), make the buffer 8 bytes larger. This is necessary because LVGL reserves 2 x 4 bytes in the buffer, as these are assumed to be used as a palette.

To skip the palette, include the following line in your *Flush Callback* function: `px_map += 8`.

As usual, monochrome displays support partial, full, and direct rendering modes as well. In full and direct modes, the buffer size should be large enough for the whole screen, meaning `(horizontal_resolution x vertical_resolution / 8) + 8` bytes. As LVGL can not handle fractional width make sure to round the horizontal resolution to 8 bits (for example 90 to 96).

The *`lv_draw_sw_i1_convert_to_vtiled()`* function is used to convert a draw buffer in I1 color format from a row-wise (htiled) to a column-wise (vtiled) buffer layout. This conversion is necessary for certain display controllers that require a different draw buffer mapping. The function assumes that the buffer width and height are rounded to a multiple of 8. The bit order of the resulting vtiled buffer can be specified using the `bit_order_lsb` parameter.

For more details, refer to the implementation in *`lv_draw_sw_i1_convert_to_vtiled()`* in `src/draw/sw/lv_draw_sw.c`.

To ensure that the redrawn areas start and end on byte boundaries, you can add a rounder callback to your display driver. This callback will round the width and height to the nearest multiple of 8.

Here is an example of how to implement and set a rounder callback:

```
static void my_rounder_cb(lv_event_t *e)
{
    lv_area_t *area = lv_event_get_param(e);

    /* Round the height to the nearest multiple of 8 */
    area->y1 = (area->y1 & ~0x7);
    area->y2 = (area->y2 | 0x7);
}

lv_display_add_event_cb(display, my_rounder_cb, LV_EVENT_INVALIDATE_AREA,␣
↪display);
```

In this example, the `my_rounder_cb` function rounds the coordinates of the redrawn area to the nearest multiple of 8. The `x1` and `y1` coordinates are rounded down, while the `x2` and `y2` coordinates are rounded up. This ensures that the width and height of the redrawn area are always multiples of 8.

Transparent Screens

Usually, the opacity of the Screen is *`LV_OPA_COVER`* to provide a solid background for its children. If this is not the case (opacity < 100%) the display's `bottom_layer` will be visible. If the bottom layer's opacity is also not *`LV_OPA_COVER`* LVGL will have no solid background to draw.

This configuration (transparent Screen) could be useful to create, for example, on-screen display (OSD) menus where a video is played on a different hardware layer of the display panel, and a menu is overlaid on a higher layer.

To properly render a UI on a transparent Screen the Display's color format needs to be set to one with an alpha channel (for example LV_COLOR_FORMAT_ARGB8888).

In summary, to enable transparent screens and displays for OSD menu-like UIs:

- Set the screen's `bg_opa` to transparent: `lv_obj_set_style_bg_opa(lv_screen_active(), LV_OPA_TRANSP, LV_PART_MAIN)`
- Set the bottom layer's `bg_opa` to transparent: `lv_obj_set_style_bg_opa(lv_layer_bottom(), LV_OPA_TRANSP, LV_PART_MAIN)`
- Set a color format with alpha channel. E.g. `lv_display_set_color_format(disp, LV_COLOR_FORMAT_ARGB8888)`

API

lv_conf_internal.h

lv_display.h

lv_draw_sw_utils.h

lv_obj_style_gen.h

7.1.5 Refreshing

Default Refresh Behavior

Normally the dirty (a.k.a invalid) areas are checked and redrawn in every `LV_DEF_REFR_PERIOD` milliseconds (set in `lv_conf.h`). This happens as a result of a refresh *Timer (lv_timer)* created that gets created when the display is created, and is executed at that interval.

Decoupling the Display Refresh Timer

However, in some cases you might need more control on when display refreshing happens, for example:

- to synchronize rendering with VSYNC or the TE signal;
- to time display refreshes immediately after a single screen update of all widgets that needed to have their display data updated (i.e. only updated once immediately before display refresh to reduce CPU overhead).

You can do this in the following way:

```
/* Delete original display refresh timer */
lv_display_delete_refr_timer(display1);

/* Call this to refresh dirty (changed) areas of the display. */
lv_display_refr_timer(NULL);
```

If you have multiple displays call `lv_display_set_default(display1)` to select the display to refresh before `lv_display_refr_timer(NULL)`.

> **Nota**
>
> `lv_timer_handler()` and `lv_display_refr_timer()` must not run at the same time.

If the performance monitor is enabled, the value of `LV_DEF_REFR_PERIOD` needs to be set to match the refresh period of the display to ensure that the statistical results are correct.

Forcing a Refresh

Normally the invalidated areas (marked for redrawing) are rendered in `lv_timer_handler()` in every `LV_DEF_REFR_PERIOD` milliseconds. However, by using `lv_refr_now(display)` you can tell LVGL to redraw the invalid areas immediately. The refreshing will happen in `lv_refr_now()` which might take longer.

The parameter of `lv_refr_now()` is a pointer to the display to refresh. If `NULL` is passed, all displays that have active refresh timers will be refreshed.

API

lv_api_map_v9_1.h

lv_conf_internal.h

lv_display.h

lv_refr.h

lv_timer.h

7.1.6 Display Events

`lv_display_add_event_cb(disp, event_cb, LV_EVENT_..., user_data)` adds an event handler to a display.

If you added `user_data` to the Display, you can retrieve it in an event like this:

```
lv_display_t * display1;
my_type_t    * my_user_data;
display1 = (lv_display_t *)lv_event_get_current_target(e);
my_user_data = lv_display_get_user_data(display1);
```

The following events are sent for Display (lv_display_t) objects:

- `LV_EVENT_INVALIDATE_AREA` An area is invalidated (marked for redraw). `lv_event_get_param(e)` returns a pointer to an `lv_area_t` object with the coordinates of the area to be invalidated. The area can be freely modified if needed to adapt it a special requirement of the display. Usually needed with monochrome displays to invalidate `N x 8` rows or columns in one pass.
- `LV_EVENT_RESOLUTION_CHANGED`: Sent when the resolution changes due to `lv_display_set_resolution()` or `lv_display_set_rotation()`.
- `LV_EVENT_COLOR_FORMAT_CHANGED`: Sent as a result of any call to `lv_display_set_color_format()`.
- `LV_EVENT_REFR_REQUEST`: Sent when something happened that requires redraw.
- `LV_EVENT_REFR_START`: Sent before a refreshing cycle starts. Sent even if there is nothing to redraw.
- `LV_EVENT_REFR_READY`: Sent when refreshing has been completed (after rendering and calling *Flush Callback*). Sent even if no redraw happened.
- `LV_EVENT_RENDER_START`: Sent just before rendering begins.
- `LV_EVENT_RENDER_READY`: Sent after rendering has been completed (before calling *Flush Callback*)
- `LV_EVENT_FLUSH_START`: Sent before *Flush Callback* is called.
- `LV_EVENT_FLUSH_FINISH`: Sent after *Flush Callback* call has returned.
- `LV_EVENT_FLUSH_WAIT_START`: Sent at the beginning of internal call to `wait_for_flushing()` -- happens whether or not any waiting will actually occur. Call returns immediately if `disp->flushing == 0`.
- `LV_EVENT_FLUSH_WAIT_FINISH`: Sent when the call to `wait_for_flushing()` is about to return, regardless whether any actual waiting occurred.

API

lv_display.h

7.1.7 Changing Resolution

To set the resolution of the display after creation use `lv_display_set_resolution(display, hor_res, ver_res)`

It's not mandatory to use the whole display for LVGL, however in some cases the physical resolution is important. For example, the touchpad still sees the whole resolution, and the values need to be converted to the active LVGL display area. So the physical resolution and the offset of the active area can be set with `lv_display_set_physical_resolution(disp, hor_res, ver_res)` and `lv_display_set_offset(disp, x, y)`

API

lv_display.h

7.1.8 Inactivity Measurement

A user's inactivity time is measured and stored with each `lv_display_t` object. Every use of an *Input Device* (if *associated with the display*) counts as activity. To get time elapsed since the last activity, use `lv_display_get_inactive_time(display1)`. If `NULL` is passed, the lowest inactivity time among all displays will be returned (in this case NULL does *not* mean the *Display di Default*).

You can manually trigger an activity using `lv_display_trigger_activity(display1)`. If `display1` is `NULL`, the *Display di Default* will be used (**not all displays**).

Further Reading

- lv_port_disp_template.c for a template for your own driver.
- *Drawing* to learn more about how rendering works in LVGL.

API

lv_display.h

7.1.9 Rotation

LVGL supports rotation of the display in 90 degree increments.

The orientation of the display can be changed with `lv_display_set_rotation(disp, LV_DISPLAY_ROTATION_xxx)` where `xxx` is 0, 90, 180 or 270. This will swap the horizontal and vertical resolutions internally according to the set degree; however, it will not perform the actual rotation. When changing the rotation, the `LV_EVENT_SIZE_CHANGED` event is emitted to allow for hardware reconfiguration. If your display panel and/or its driver chip(s) do not support rotation, `lv_draw_sw_rotate()` can be used to rotate the buffer in the *Flush Callback* function.

`lv_display_rotate_area(display, &area)` rotates the rendered area according to the current rotation settings of the display.

Note that in `LV_DISPLAY_RENDER_MODE_DIRECT` the small changed areas are rendered directly in the frame buffer so they cannot be rotated later. Therefore in direct mode only the whole frame buffer can be rotated.

In the case of `LV_DISPLAY_RENDER_MODE_PARTIAL` the small rendered areas can be rotated on their own before flushing to the frame buffer.

`LV_DISPLAY_RENDER_MODE_FULL` can work with rotation if the buffer(s) being rendered to are different than the buffer(s) being rotated to in the flush callback and the buffers being rendered to do not have a stride requirement.

Below is an example for rotating when the rendering mode is *LV_DISPLAY_RENDER_MODE_PARTIAL* and the rotated image should be sent to a **display controller**.

```
/*Rotate a partially rendered area to another buffer and send it*/
void flush_cb(lv_display_t * disp, const lv_area_t * area, uint8_t * px_
↪map)
{
    lv_display_rotation_t rotation = lv_display_get_rotation(disp);
    lv_area_t rotated_area;
    if(rotation != LV_DISPLAY_ROTATION_0) {
        lv_color_format_t cf = lv_display_get_color_format(disp);
        /*Calculate the position of the rotated area*/
        rotated_area = *area;
        lv_display_rotate_area(disp, &rotated_area);
        /*Calculate the source stride (bytes in a line) from the width of␣
↪the area*/
        uint32_t src_stride = lv_draw_buf_width_to_stride(lv_area_get_
↪width(area), cf);
        /*Calculate the stride of the destination (rotated) area too*/
        uint32_t dest_stride = lv_draw_buf_width_to_stride(lv_area_get_
↪width(&rotated_area), cf);
        /*Have a buffer to store the rotated area and perform the␣
↪rotation*/
        static uint8_t rotated_buf[500*1014];
        int32_t src_w = lv_area_get_width(area);
        int32_t src_h = lv_area_get_height(area);
        lv_draw_sw_rotate(px_map, rotated_buf, src_w, src_h, src_stride,␣
↪dest_stride, rotation, cf);
        /*Use the rotated area and rotated buffer from now on*/
        area = &rotated_area;
        px_map = rotated_buf;
    }
    my_set_window(area->x1, area->y1, area->x2, area->y2);
    my_send_colors(px_map);
}
```

Below is an example for rotating when the rendering mode is *LV_DISPLAY_RENDER_MODE_PARTIAL* and the image can be rotated directly into a **frame buffer of the LCD peripheral**.

```
/*Rotate a partially rendered area to the frame buffer*/
void flush_cb(lv_display_t * disp, const lv_area_t * area, uint8_t * px_
↪map)
{
    lv_color_format_t cf = lv_display_get_color_format(disp);
    uint32_t px_size = lv_color_format_get_size(cf);
    /*Calculate the position of the rotated area*/
    lv_area_t rotated_area = *area;
    lv_display_rotate_area(disp, &rotated_area);
    /*Calculate the properties of the source buffer*/
    int32_t src_w = lv_area_get_width(area);
    int32_t src_h = lv_area_get_height(area);
    uint32_t src_stride = lv_draw_buf_width_to_stride(src_w, cf);
    /*Calculate the properties of the frame buffer*/
    int32_t fb_stride = lv_draw_buf_width_to_stride(disp->hor_res, cf);
    uint8_t * fb_start = my_fb_address;
    fb_start += rotated_area.y1 * fb_stride + rotated_area.x1 * px_size;
    lv_display_rotation_t rotation = lv_display_get_rotation(disp);
    if(rotation == LV_DISPLAY_ROTATION_0) {
```

(continues on next page)

(continua dalla pagina precedente)

```
        int32_t y;
        for(y = area->y1; y <= area->y2; y++) {
            lv_memcpy(fb_start, px_map, src_stride);
            px_map += src_stride;
            fb_start += fb_stride;
        }
    }
    else {
        lv_draw_sw_rotate(px_map, fb_start, src_w, src_h, src_stride, fb_
↪stride, rotation, cf);
    }
}
```

API

lv_display.h

7.1.10 Constraints on Redrawn Area

Some display controllers have specific requirements for the window area where the rendered image can be sent (e.g., `x1` must be even, and `x2` must be odd).

In the case of monochrome displays, `x1` must be `Nx8`, and `x2` must be `Nx8 - 1`. (If the display uses `LV_COLOR_FORMAT_I1`, LVGL automatically applies this rounding. See *Monochrome Displays*.)

The size of the invalidated (redrawn) area can be controlled as follows:

```
void rounder_event_cb(lv_event_t * e)
{
    lv_area_t * a = lv_event_get_invalidated_area(e);

    a->x1 = a->x1 & (~0x1); /* Ensure x1 is even */
    a->x2 = a->x2 | 0x1;    /* Ensure x2 is odd */
}

...

lv_display_add_event_cb(disp, rounder_event_cb, LV_EVENT_INVALIDATE_AREA,␣
↪NULL);
```

API

lv_display.h

7.1.11 Tiled Rendering

When multiple CPU cores are available and a large area needs to be redrawn, LVGL must identify independent areas that can be rendered in parallel.

For example, if there are 4 CPU cores, one core can draw the screen's background while the other 3 must wait until it is finished. If there are 2 buttons on the screen, those 2 buttons can be rendered in parallel, but 2 cores will still remain idle.

Due to dependencies among different areas, CPU cores cannot always be fully utilized.

To address this, LVGL can divide large areas that need to be updated into smaller tiles. These tiles are independent, making it easier to find areas that can be rendered concurrently.

Specifically, if there are 4 tiles and 4 cores, there will always be an independent area for each core within one of the tiles.

The maximum number of tiles can be set using the function `lv_display_set_tile_cnt(disp, cnt)`. The default value is `LV_DRAW_SW_DRAW_UNIT_CNT` (or 1 if software rendering is not enabled).

Small areas are not further divided into smaller tiles because the overhead of spinning up 4 cores would outweigh the benefits.

The ideal tile size is calculated as `ideal_tile_size = draw_buf_size / tile_cnt`. For example, in `LV_DISPLAY_RENDER_MODE_DIRECT` mode on an 800x480 screen, the display buffer is 800x480 = 375k pixels. If there are 4 tiles, the ideal tile size is approximately 93k pixels. Based on this, core utilization is as follows:

- 30k pixels: 1 core
- 90k pixels: 1 core
- 95k pixels: 2 cores (above 93k pixels, 2 cores are used)
- 150k pixels: 2 cores
- 200k pixels: 3 cores (above 186k pixels, 3 cores are used)
- 300k pixels: 4 cores (above 279k pixels, 4 cores are used)
- 375k pixels: 4 cores

In `LV_DISPLAY_RENDER_MODE_DIRECT`, the screen-sized draw buffer is divided by the tile count to determine the ideal tile sizes. If smaller areas are refreshed, it may result in fewer cores being used.

In `LV_DISPLAY_RENDER_MODE_FULL`, the maximum number of tiles is always created when the entire screen is refreshed.

In `LV_DISPLAY_RENDER_MODE_PARTIAL`, the partial buffer is divided into tiles. For example, if the draw buffer is 1/10th the size of the screen and there are 2 tiles, then 1/20th + 1/20th of the screen area will be rendered at once.

Tiled rendering only affects the rendering process, and the *Flush Callback* is called once for each invalidated area. Therefore, tiling is not visible from the flushing point of view.

API

lv_display.h

7.1.12 Extending/Combining Displays

Mirroring a Display

To mirror the image of a display to another display, you don't need to use multi-display support. Just transfer the buffer received in the first display's *Flush Callback* to the other display as well.

Splitting an Image

You can create a larger virtual display from an array of smaller ones. You can create it by:

1. setting the resolution of the displays to the large display's resolution;
2. in *Flush Callback*, truncate and modify the `area` parameter for each display; and
3. send the buffer's content to each real display with the truncated area.

7.1.13 API

lv_display.h

7.2 Input Device (lv_indev)

7.2.1 Creating an Input Device

To create an input device on the *Display di Default*:

```
/* Create and set up at least one display before you register any input␣
↪devices. */
lv_indev_t * indev = lv_indev_create();        /* Create input device␣
↪connected to Default Display. */
lv_indev_set_type(indev, LV_INDEV_TYPE_...);   /* Touch pad is a pointer-
↪like device. */
lv_indev_set_read_cb(indev, my_input_read);    /* Set driver function. */
```

If you have multiple displays, you will need to ensure the Default Display is set to the display your input device is "connected to" before making the above calls.

The `type` member can be:

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

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

Touchpad, Touch-Screen, Mouse or Any Pointer

Input devices that can click points on the display belong to the POINTER category. Here is an example of a simple input-device Read Callback function:

```
lv_indev_set_type(indev, LV_INDEV_TYPE_POINTER);
...

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

Mouse Cursor

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

```
...
lv_indev_t * mouse_indev = lv_indev_create();
...
LV_IMAGE_DECLARE(mouse_cursor_icon);                        /* Declare␣
↪the image source. */
lv_obj_t * cursor_obj = lv_image_create(lv_screen_active());  /* Create␣
↪image Widget for cursor. */
lv_image_set_src(cursor_obj, &mouse_cursor_icon);           /* Set image␣
```

(continues on next page)

(continua dalla pagina precedente)

```
↪source. */
lv_indev_set_cursor(mouse_indev, cursor_obj);                /* Connect␣
↪image to Input Device. */
```

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

Gestures

Pointer input devices can detect basic gestures. By default, most Widgets send gestures to their parents so they can be detected on the Screen Widget in the form of an `LV_EVENT_GESTURE` event. Ad esempio:

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

...

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

To prevent passing the gesture event to the parent from a Widget, use `lv_obj_remove_flag(widget, LV_OBJ_FLAG_GESTURE_BUBBLE)`.

Note that, gestures are not triggered if a Widget is being scrolled.

If you did some action on a gesture you can call `lv_indev_wait_release(lv_indev_active())` in the event handler to prevent LVGL sending further input-device-related events.

Crown Behavior

A "Crown" is a rotary device typically found on smart watches.

When the user clicks somewhere and after that turns the rotary the last clicked widget will be either scrolled or it's value will be incremented/decremented (e.g. in case of a slider).

As this behavior is tightly related to the last clicked widget, the crown support is an extension of the pointer input device. Just set `data->diff` to the number of turned steps and LVGL will automatically send the `LV_EVENT_ROTARY` event or scroll the widget based on the `editable` flag in the widget's class. Non-editable widgets are scrolled and for editable widgets the event is sent.

To get the steps in an event callback use `int32_t diff = lv_event_get_rotary_diff(e)`

The rotary sensitivity can be adjusted on 2 levels:

1. in the input device by the `indev->rotary_sensitivity` element (1/256 unit), and

2. by the `rotary_sensitivity` style property in the widget (1/256 unit).

The final diff is calculated like this:

```
diff_final = diff_in * (indev_sensitivity / 256) + (widget_sensitivity /
256);
```

For example, if both the indev and widget sensitivity is set to 128 (0.5), the input diff will be multiplied by 0.25. The value of the Widget will be incremented by that value or the Widget will be scrolled that amount of pixels.

Multi-touch gestures

LVGL has the ability to recognize multi-touch gestures, when a gesture is detected a `LV_EVENT_GESTURE` is passed to the object on which the gesture occurred. Currently, these multi-touch gestures are supported:

- Two fingers pinch (up and down)
- Two fingers rotation
- Two fingers swipe (infinite)

To enable the multi-touch gesture recognition set the `LV_USE_GESTURE_RECOGNITION` option in the `lv_conf.h` file.

Currently, the system sends the events if the gestures are in one of the following states:

- `LV_INDEV_GESTURE_STATE_RECOGNIZED`: The gesture has been recognized and is now active.
- `LV_INDEV_GESTURE_STATE_ENDED`: The gesture has ended.

Multi-touch gestures overview

To recognize multi touch gestures, recognizers are used. The structure `lv_indev_t` contains an array of recognizers, one per gesture type. These recognizers are initialized internally by `lv_indev_create` by calling `lv_indev_gesture_init_recognizers` after the indev device is created. The the recognizers can then be configured to modify the gestures thresholds. These thresholds are used to be able to recognize the gesture only after the threshold have been reached. They can be set-up like this:

- `lv_indev_set_pinch_up_threshold(lv_indev_t * indev, float threshold)`: Set the pinch up (zoom in) threshold in pixels.
- `lv_indev_set_pinch_down_threshold(lv_indev_t * indev, float threshold)`: Set the pinch down (zoom out) threshold in pixels.
- `lv_indev_set_rotation_rad_threshold(lv_indev_t * indev, float threshold)`: Set the rotation angle threshold in radians.

The recognizers can then be updated to recognize the gestures by calling `lv_indev_gesture_recognizers_update`. This must be done in the user defined indev `read_cb`. This will iterate over the recognizers and stop once it detects a recognized or ended gesture. For now only one multi-touch gesture can be recognized/ended at a time.

Once the recognizers are updated, calling `lv_indev_gesture_recognizers_set_data` will update the `lv_indev_data_t` structure. It is meant to be done in the indev `read_cb`. This allows the future `lv_event_t` to eb filled with multi-touch gesture info.

Here is an example of the `read_cb`:

```
/* The recognizer keeps the state of the gesture */
static lv_indev_gesture_recognizer_t recognizer;

/* An array that stores the collected touch events */
static lv_indev_touch_data_t touches[10];

/* A counter that needs to be incremented each time a touch event is␣
```

(continues on next page)

(continua dalla pagina precedente)

```
↪received */
static uint8_t touch_cnt;

static void touch_read_callback(lv_indev_t * drv, lv_indev_data_t * data)
{

     lv_indev_touch_data_t * touch;

     lv_indev_update_recognizers(drv, &touches[0], touch_cnt);

     touch_cnt = 0;

     /* Set the gesture information, before returning to LVGL */
     lv_indev_gesture_recognizers_set_data(drv, data);

}
```

The user is in charge of collecting the necessary touches events from the driver until the indev `read_cb` is called. It must then convert the specific driver input to `lv_indev_touch_data_t` to be processed by the `read_cb` at a later point. Here is an example using `libinput`:

```
/**
* @brief Convert the libinput to lvgl's representation of touch event
* @param ev a pointer to the lib input event
*/
static void touch_event_queue_add(struct libinput_event *ev)
{
    struct libinput_event_touch *touch_ev;
    lv_indev_touch_data_t *cur;
    lv_indev_touch_data_t *t;
    uint32_t time;
    int i;
    int id;
    int type;

    type = libinput_event_get_type(ev);
    touch_ev = libinput_event_get_touch_event(ev);
    id = libinput_event_touch_get_slot(touch_ev);
    time = libinput_event_touch_get_time(touch_ev);

    /* Get the last event for contact point */
    t = &touches[0];
    cur = NULL;

    for (i = 0; i < touch_cnt; i++) {
        if (t->id == id) {
            cur = t;
        }
        t++;
    }

    if (cur != NULL && cur->timestamp == time) {
        /* Previous event has the same timestamp - ignore duplicate event␣
↪*/
        return;
    }
```

(continues on next page)

(continua dalla pagina precedente)

```
    if (cur == NULL ||
            type == LIBINPUT_EVENT_TOUCH_UP ||
            type == LIBINPUT_EVENT_TOUCH_DOWN) {

        /* create new event */
        cur = &touches[touch_cnt];
        touch_cnt++;
    }

    switch (type) {
        case LIBINPUT_EVENT_TOUCH_DOWN:
        case LIBINPUT_EVENT_TOUCH_MOTION:

            cur->point.x = (int) libinput_event_touch_get_x_
↪transformed(touch_ev, SCREEN_WIDTH);
            cur->point.y = (int) libinput_event_touch_get_y_
↪transformed(touch_ev, SCREEN_HEIGHT);
            cur->state = LV_INDEV_STATE_PRESSED;
            break;

        case LIBINPUT_EVENT_TOUCH_UP:

            cur->state = LV_INDEV_STATE_RELEASED;
            cur->point.x = 0;
            cur->point.y = 0;
            break;
    }

    cur->timestamp = time;
    cur->id = id;
}

/**
* @brief Filter out libinput events that are not related to touches
* @param ev a pointer to the lib input event
*/
static void process_libinput_event(struct libinput_event *ev)
{
    int type;

    type = libinput_event_get_type(ev);

    switch (type) {
        case LIBINPUT_EVENT_TOUCH_MOTION:
        case LIBINPUT_EVENT_TOUCH_DOWN:
        case LIBINPUT_EVENT_TOUCH_UP:
            /* Filter only touch events */
            touch_event_queue_add(ev);
            break;
        default:
            /* Skip an unrelated libinput event */
            return;
    }
}
```

From this setup, the user can now register events callbacks to react to `LV_EVENT_GESTURE`.

Nota

A touch event is represented by the `lv_indev_touch_data_t` structure, the fields being 1:1 compatible with events emitted by the libinput library

Handling multi-touch gesture events

Once a gesture is recognized or ended, a `LV_EVENT_GESTURE` is sent. The user can the use these functions to gather more information about the gesture:

- `lv_event_get_gesture_type(lv_event_t * gesture_event)`: Get the type of the gesture. To be used to check which multi-touch gesture is currently reported.
- `lv_indev_gesture_state_t lv_event_get_gesture_state(lv_event_t * gesture_event, lv_indev_gesture_type_t type)`: Get the state of the gesture. It can be one of those:
  - `LV_INDEV_GESTURE_STATE_NONE`: The gesture is not active.
  - `LV_INDEV_GESTURE_STATE_RECOGNIZED`: The gesture is recognized and can be used.
  - `LV_INDEV_GESTURE_STATE_ENDED`: The gesture ended.

These functions allow the user to confirm the gesture is the expected one and that it is in a usable state. The user can then request the gestures values with the following functions:

- `lv_event_get_pinch_scale(lv_event_t * gesture_event)`: Get the pinch scale. Only relevant for pinch gesture.
- `lv_event_get_rotation(lv_event_t * gesture_event)`: Get the rotation in radians. Only relevant for rotation gesture.
- `lv_event_get_two_fingers_swipe_distance(lv_event_t * gesture_event)`: Get the distance in pixels from the gesture staring center. Only relevant for two fingers swipe gesture.
- `lv_event_get_two_fingers_swipe_dir(lv_event_t * gesture_event)`: Get the direction from the starting center. Only relevant for two fingers swipe gesture.

This allow the user to react to the gestures and to use the gestures values. An example of such an application is available in the source tree `examples/others/gestures/lv_example_gestures.c`.

Keypad or Keyboard

Full keyboards with all the letters or simple keypads with a few navigation buttons belong in the keypad category.

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

To use a keyboard or keypad:

- Register a Read Callback function for your device and set its type to *`LV_INDEV_TYPE_KEYPAD`*.
- Create a Widget Group (`lv_group_t * g = lv_group_create()`) and add Widgets to it with *`lv_group_add_obj`*`(g, widget)`.
- Assign the group to an input device: *`lv_indev_set_group`*`(indev, g)`.
- Use `LV_KEY_...` to navigate among the Widgets in the group. See `lv_core/lv_group.h` for the available keys.

```
lv_indev_set_type(indev, LV_INDEV_TYPE_KEYPAD);

...

void keyboard_read(lv_indev_t * indev, lv_indev_data_t * data){
```

(continues on next page)

(continua dalla pagina precedente)

```
  data->key = last_key();            /* Get the last pressed or released␣
↪key */

  if(key_pressed()) data->state = LV_INDEV_STATE_PRESSED;
  else data->state = LV_INDEV_STATE_RELEASED;
}
```

Encoder

A common example of an encoder is a device with a turning knob that tells the hosting device *when* the knob is being turned, and *in which direction*.

With an encoder your application can receive events from the following:

1. press of its button,
2. oong-press of its button,
3. turn left, and
4. turn right.

In short, the Encoder input devices work like this:

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

To use an Encoder (similar to the *Keypads*) the Widgets should be added to a group.

```
lv_indev_set_type(indev, LV_INDEV_TYPE_ENCODER);

...

void encoder_read(lv_indev_t * indev, lv_indev_data_t * data){
  data->enc_diff = enc_get_new_moves();

  if(enc_pressed()) data->state = LV_INDEV_STATE_PRESSED;
  else data->state = LV_INDEV_STATE_RELEASED;
}
```

Widget Groups

When input focus needs to be managed among a set of Widgets (e.g. to capture user input from a keypad or encoder), that set of Widgets is placed in a group which thereafter manages how input focus moves from Widget to Widget.

In each group there is exactly one object with focus which receives the pressed keys or the encoder actions. For example, if a *Text Area* has focus and you press some letter on a keyboard, the keys will be sent and inserted into the text area. Similarly, if a *Slider* has focus and you press the left or right arrows, the slider's value will be changed.

You need to associate an input device with a group. An input device can send key events to only one group but a group can receive data from more than one input device.

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

Once a Widget has been added to a group, you can find out what group it is in using `lv_obj_get_group(widget)`.

To find out what Widget in a group has focus, if any, call `lv_group_get_focused(group)`. If a Widget in that group has focus, it will return a pointer to it, otherwise it will return NULL.

To associate a group with an input device use `lv_indev_set_group(indev, g)`.

Keys

There are some predefined keys which have special meaning:

- `LV_KEY_NEXT`: Move focus to next object
- `LV_KEY_PREV`: Move focus to previous object
- `LV_KEY_ENTER`: Triggers `LV_EVENT_PRESSED`, `LV_EVENT_CLICKED`, or `LV_EVENT_LONG_PRESSED` etc. events
- `LV_KEY_UP`: Increase value or move up
- `LV_KEY_DOWN`: Decrease value or move down
- `LV_KEY_RIGHT`: Increase value or move to the right
- `LV_KEY_LEFT`: Decrease value or move to the left
- `LV_KEY_ESC`: Close or exit (e.g. close a *Drop-Down List*)
- `LV_KEY_DEL`: Delete (e.g. a character on the right in a *Text Area*)
- `LV_KEY_BACKSPACE`: Delete (e.g. a character on the left in a *Text Area*)
- `LV_KEY_HOME`: Go to the beginning/top (e.g. in a *Text Area*)
- `LV_KEY_END`: Go to the end (e.g. in a *Text Area*)

The most important special keys in your `read_cb()` function are:

- `LV_KEY_NEXT`
- `LV_KEY_PREV`
- `LV_KEY_ENTER`
- `LV_KEY_UP`
- `LV_KEY_DOWN`
- `LV_KEY_LEFT`
- `LV_KEY_RIGHT`

You should translate some of your keys to these special keys to support navigation in a group and interact with selected Widgets.

Usually, it's enough to use only `LV_KEY_LEFT` and `LV_KEY_RIGHT` because most Widgets can be fully controlled with them.

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

Edit and Navigate Mode

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

In *Navigate* mode, an encoder's `LV_KEY_LEFT` or `LV_KEY_RIGHT` is translated to `LV_KEY_NEXT` or `LV_KEY_PREV`. Therefore, the next or previous object will be selected by turning the encoder. Pressing `LV_KEY_ENTER` will change to *Edit* mode.

In *Edit* mode, `LV_KEY_NEXT` and `LV_KEY_PREV` is usually used to modify an object. Depending on the Widget's type, a short or long press of `LV_KEY_ENTER` changes back to *Navigate* mode. Usually, a Widget which cannot be

pressed (like a *Slider*) leaves *Edit* mode upon a short click. But with Widgets where a short click has meaning (e.g. *Button*), a long press is required.

Default Group

Interactive widgets (such as Buttons, Checkboxes, Sliders, etc.) can be automatically added to a default group. Just create a group with `lv_group_t * g = lv_group_create()` and set the default group with `lv_group_set_default(g)`

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

Styling

When a Widget receives focus either by clicking it via touchpad or by navigating to it with an encoder or keypad, it goes to the `LV_STATE_FOCUSED` state. Hence, focused styles will be applied to it.

If a Widget switches to edit mode it enters the `LV_STATE_FOCUSED | LV_STATE_EDITED` states so any style properties assigned to these states will be shown.

See *Stili* for more details.

Using Buttons with Encoder Logic

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

You need to have 3 buttons available:

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

If you hold the keys it will simulate an encoder advance with period specified in `indev_drv.long_press_repeat_time`.

```
lv_indev_set_type(indev, LV_INDEV_TYPE_ENCODER);

...

void encoder_with_keys_read(lv_indev_t * indev, lv_indev_data_t * data){
  data->key = last_key();            /* Get the last pressed or released␣
↪key */
                                     /* use LV_KEY_ENTER for encoder press␣
↪*/
  if(key_pressed()) data->state = LV_INDEV_STATE_PRESSED;
  else {
      data->state = LV_INDEV_STATE_RELEASED;
      /* Optionally you can also use enc_diff, if you have encoder */
      data->enc_diff = enc_get_new_moves();
  }
}
```

Hardware Button

A *Hardware Button* here is an external button (switch) typically next to the screen which is assigned to specific coordinates of the screen. If a button is pressed it will simulate the pressing on the assigned coordinate, similar to a touchpad.

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

> **Important**
>
> `points_array` cannot be allowed to go out of scope. Either declare it as a global variable or as a static variable inside a function.

```
lv_indev_set_type(indev, LV_INDEV_TYPE_BUTTON);

...

void button_read(lv_indev_t * indev, lv_indev_data_t * data){
    static uint32_t last_btn = 0;   /* Store the last pressed button */
    int btn_pr = my_btn_read();     /* Get the ID (0,1,2...) of the␣
↪pressed button */
    if(btn_pr >= 0) {               /* Is there a button press? (E.g. -1␣
↪indicated no button was pressed) */
       last_btn = btn_pr;           /* Save the ID of the pressed button */
       data->state = LV_INDEV_STATE_PRESSED;  /* Set the pressed state */
    } else {
       data->state = LV_INDEV_STATE_RELEASED; /* Set the released state */
    }

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

When the `button_read` callback in the example above changes the `data->btn_id` to `0` a press/release action at the first index of the `points_array` will be performed (`{12,30}`).

7.2.2 Other Features

Parameters

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

- `scroll_limit` Number of pixels to slide before actually scrolling the Widget
- `scroll_throw` Scroll throw (momentum) slow-down in [%]. Greater value means faster slow-down.
- `long_press_time` Press time to send *`LV_EVENT_LONG_PRESSED`* (in milliseconds)
- `long_press_repeat_time` Interval of sending *`LV_EVENT_LONG_PRESSED_REPEAT`* (in milliseconds)
- `read_timer` pointer to the `lv_timer` which reads the input device. Its parameters can be changed by calling `lv_timer_...()` functions. *`LV_DEF_REFR_PERIOD`* in `lv_conf.h` sets the default read period.

Buffered Reading

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

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

If the driver can provide precise timestamps for buffered events, it can overwrite `data->timestamp`. By default, this is initialized to *`lv_tick_get()`* just before invoking `read_cb`.

Switching the Input Device to Event-Driven Mode

Normally an Input Device is read every *`LV_DEF_REFR_PERIOD`* milliseconds (set in `lv_conf.h`). However, in some cases, you might need more control over when to read the input device. For example, you might need to read it by polling a file descriptor (fd).

You can do this by:

```
/* Update the input device's running mode to LV_INDEV_MODE_EVENT */
lv_indev_set_mode(indev, LV_INDEV_MODE_EVENT);

...

/* Call this anywhere you want to read the input device */
lv_indev_read(indev);
```

Nota

`lv_indev_read()`, *`lv_timer_handler()`* and `_lv_display_refr_timer()` cannot run at the same time.

Nota

For devices in event-driven mode, `data->continue_reading` is ignored.

Further Reading

- lv_port_indev_template.c for a template for your own Input-Device driver.

7.2.3 API

lv_api_map_v8.h

lv_api_map_v9_1.h

lv_indev.h

lv_indev_gesture.h

lv_indev_gesture_private.h

lv_indev_private.h

lv_indev_scroll.h

7.3 Color (lv_color)

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

The type *lv_color_t* is used to store a color in RGB888 format. This type and format are used in almost all APIs regardless of LV_COLOR_DEPTH.

7.3.1 Creating Colors

RGB

Create colors from Red, Green and Blue channel values:

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


/* Same but can be used for const initialization as well */
lv_color_t c = LV_COLOR_MAKE(red, green, blue);

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

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

HSV

Create colors from Hue, Saturation and Value values:

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

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


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

Palette

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

The names of the colors are as follows:

- LV_PALETTE_RED
- LV_PALETTE_PINK
- LV_PALETTE_PURPLE
- LV_PALETTE_DEEP_PURPLE
- LV_PALETTE_INDIGO
- LV_PALETTE_BLUE
- LV_PALETTE_LIGHT_BLUE
- LV_PALETTE_CYAN

- LV_PALETTE_TEAL
- LV_PALETTE_GREEN
- LV_PALETTE_LIGHT_GREEN
- LV_PALETTE_LIME
- LV_PALETTE_YELLOW
- LV_PALETTE_AMBER
- LV_PALETTE_ORANGE
- LV_PALETTE_DEEP_ORANGE
- LV_PALETTE_BROWN
- LV_PALETTE_BLUE_GREY
- LV_PALETTE_GREY

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

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

Modify and mix colors

The following functions can modify a color:

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

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

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


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

Built-in colors

lv_color_white() and *lv_color_black()* return 0xFFFFFF and 0x000000 respectively.

7.3.2 Opacity

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

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

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

7.3.3 API

lv_color.h

lv_color_op.h

lv_draw_ppa_private.h

7.4 Font (lv_font)

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

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

Fonts have a **format** property. It describes how the glyph data is stored. At the time of writing, there are 12 possible values that this field can take, and those values fall into 2 categories:

Legacy simple
: 1, 2, 4 or 8-bpp (aligned or unaligned) and image format, and

Advanced
: vector, SVG, and custom formats; for the latter, the user provides the rendering logic.

For simple formats:

- the font is stored as an array of bitmaps, one bitmap per glyph;
- the value stored for each pixel determines the pixel's opacity, enabling edges to be smoother --- higher bpp values result in smoother edges.

For advanced formats, the font information is stored in its respective format.

The **format** property also affects the amount of memory needed to store a font. For example, `format = LV_FONT_GLYPH_FORMAT_A4` makes a font nearly four times larger compared to `format = LV_FONT_GLYPH_FORMAT_A1`.

7.4.1 Unicode Support

LVGL supports **UTF-8** encoded Unicode characters. Your editor needs to be configured to save your code/text as UTF-8 (usually this the default) and be sure that *LV_TXT_ENC* is set to *LV_TXT_ENC_UTF8* in `lv_conf.h`. (This is the default value.)

To test it try

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

If all works well, a '✓' character should be displayed.

7.4.2 Typesetting

Although LVGL can decode and display any Unicode characters (assuming the font supports them), LVGL cannot correctly render all complex languages.

The standard Latin-based languages (e.g., English, Spanish, German) and East Asian languages such as Chinese, Japanese, and Korean (CJK) are relatively straightforward, as their characters are simply written from left to right.

Languages like Arabic, Persian, and Hebrew, which use Right-to-Left (RTL) or mixed writing directions, are also supported in LVGL. Learn more *here*.

For characters such as 'é', 'ü', 'õ', 'Á', and 'Ũ', it is recommended to use the single Unicode format (NFC) rather than decomposing them into a base letter and diacritics (e.g. u + ¨).

Complex languages where subsequent characters combine into a single glyph and where the resulting glyph has no individual Unicode representation (e.g., Devanagari), have limited support in LVGL.

7.4.3 Built-In Fonts

There are several built-in fonts in different sizes, which can be enabled in `lv_conf.h` with *LV_FONT_...* defines.

Normal Fonts

The following fonts contain all ASCII characters, the degree symbol (U+00B0), the bullet symbol (U+2022) and the built-in symbols (see below).

- `LV_FONT_MONTSERRAT_12`: 12 px font
- `LV_FONT_MONTSERRAT_14`: 14 px font
- `LV_FONT_MONTSERRAT_16`: 16 px font
- `LV_FONT_MONTSERRAT_18`: 18 px font
- `LV_FONT_MONTSERRAT_20`: 20 px font
- `LV_FONT_MONTSERRAT_22`: 22 px font
- `LV_FONT_MONTSERRAT_24`: 24 px font
- `LV_FONT_MONTSERRAT_26`: 26 px font
- `LV_FONT_MONTSERRAT_28`: 28 px font
- `LV_FONT_MONTSERRAT_30`: 30 px font
- `LV_FONT_MONTSERRAT_32`: 32 px font
- `LV_FONT_MONTSERRAT_34`: 34 px font
- `LV_FONT_MONTSERRAT_36`: 36 px font
- `LV_FONT_MONTSERRAT_38`: 38 px font
- `LV_FONT_MONTSERRAT_40`: 40 px font
- `LV_FONT_MONTSERRAT_42`: 42 px font
- `LV_FONT_MONTSERRAT_44`: 44 px font
- `LV_FONT_MONTSERRAT_46`: 46 px font
- `LV_FONT_MONTSERRAT_48`: 48 px font

Special fonts

- `LV_FONT_MONTSERRAT_28_COMPRESSED`: Same as normal 28 px font but stored as a *Compressed fonts* with 3 bpp
- `LV_FONT_DEJAVU_16_PERSIAN_HEBREW`: 16 px font with normal range + Hebrew, Arabic, Persian letters and all their forms
- `LV_FONT_SOURCE_HAN_SANS_SC_16_CJK`: 16 px font with normal range plus 1000 of the most common CJK radicals
- `LV_FONT_UNSCII_8`: 8 px pixel perfect font with only ASCII characters
- `LV_FONT_UNSCII_16`: 16 px pixel perfect font with only ASCII characters

The built-in fonts are **global variables** with names like `lv_font_montserrat_16` for a 16 px height font. To use them in a style, just add a pointer to a font variable like this:

```
lv_style_set_text_font(&my_style, &lv_font_montserrat_28);
```

The built-in fonts with `bpp = 4` contain the ASCII characters and use the Montserrat font.

In addition to the ASCII range, the following symbols are also added to the built-in fonts from the FontAwesome font.

LV_SYMBOL_AUDIO	LV_SYMBOL_WARNING
LV_SYMBOL_VIDEO	LV_SYMBOL_SHUFFLE
LV_SYMBOL_LIST	LV_SYMBOL_UP
LV_SYMBOL_OK	LV_SYMBOL_DOWN
LV_SYMBOL_CLOSE	LV_SYMBOL_LOOP
LV_SYMBOL_POWER	LV_SYMBOL_DIRECTORY
LV_SYMBOL_SETTINGS	LV_SYMBOL_UPLOAD
LV_SYMBOL_TRASH	LV_SYMBOL_CALL
LV_SYMBOL_HOME	LV_SYMBOL_CUT
LV_SYMBOL_DOWNLOAD	LV_SYMBOL_COPY
LV_SYMBOL_DRIVE	LV_SYMBOL_SAVE
LV_SYMBOL_REFRESH	LV_SYMBOL_CHARGE
LV_SYMBOL_MUTE	LV_SYMBOL_PASTE
LV_SYMBOL_VOLUME_MID	LV_SYMBOL_BELL
LV_SYMBOL_VOLUME_MAX	LV_SYMBOL_KEYBOARD
LV_SYMBOL_IMAGE	LV_SYMBOL_GPS
LV_SYMBOL_EDIT	LV_SYMBOL_FILE
LV_SYMBOL_PREV	LV_SYMBOL_WIFI
LV_SYMBOL_PLAY	LV_SYMBOL_BATTERY_FULL
LV_SYMBOL_PAUSE	LV_SYMBOL_BATTERY_3
LV_SYMBOL_STOP	LV_SYMBOL_BATTERY_2
LV_SYMBOL_NEXT	LV_SYMBOL_BATTERY_1
LV_SYMBOL_EJECT	LV_SYMBOL_BATTERY_EMPTY
LV_SYMBOL_LEFT	LV_SYMBOL_USB
LV_SYMBOL_RIGHT	LV_SYMBOL_BLUETOOTH
LV_SYMBOL_PLUS	LV_SYMBOL_BACKSPACE
LV_SYMBOL_MINUS	LV_SYMBOL_SD_CARD
LV_SYMBOL_EYE_OPEN	LV_SYMBOL_NEW_LINE
LV_SYMBOL_EYE_CLOSE	

The symbols can be used singly as:

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

Or together with strings (compile time string concatenation):

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

Or more symbols together:

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

7.4.4 Special Features

Bidirectional support

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

LVGL not only supports RTL text but supports mixed (a.k.a. bidirectional, BiDi) text rendering as well. Alcuni esempi:

BiDi support is enabled by setting *LV_USE_BIDI* to a non-zero value in `lv_conf.h`.

All text has a base direction (LTR or RTL) which determines some rendering rules and the default alignment of the text (left or right). However, in LVGL, the base direction is not only applied to labels. It's a general property which can be set for every Widget. If not set then it will be inherited from the parent. This means it's enough to set the base direction of a screen and its child Widgets will inherit it.

The default base direction for screens can be set by `LV_BIDI_BASE_DIR_DEF` in `lv_conf.h` and other Widgets inherit the base direction from their parent.

To set a Widget's base direction use *lv_obj_set_style_base_dir*`(widget, base_dir, selector)`. The possible base directions are:

- *LV_BASE_DIR_LTR*: Left to Right base direction
- *LV_BASE_DIR_RTL*: Right to Left base direction
- *LV_BASE_DIR_AUTO*: Auto detect base direction

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

- Create Widgets by default on the right
- `lv_tabview`: Displays tabs from right to left
- `lv_checkbox`: Shows the box on the right
- `lv_buttonmatrix`: Orders buttons from right to left
- `lv_list`: Shows icons on the right
- `lv_dropdown`: Aligns options to the right
- The text strings in `lv_table`, `lv_buttonmatrix`, `lv_keyboard`, `lv_tabview`, `lv_dropdown`, `lv_roller` are "BiDi processed" to be displayed correctly

Arabic and Persian support

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

LVGL supports these rules if *LV_USE_ARABIC_PERSIAN_CHARS* is enabled in `lv_conf.h`.

However, there are some limitations:

- Only displaying text is supported (e.g. on labels), i.e. text inputs (e.g. Text Area) do not support this feature.

- Static text (i.e. const) is not processed. E.g. text set by `lv_label_set_text()` will be "Arabic processed" but `lv_label_set_text_static()` will not.
- Text get functions (e.g. `lv_label_get_text()`) will return the processed text.

Compressed fonts

The built-in font engine supports compressed bitmaps. Compressed fonts can be generated by

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

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

- they need the most memory
- they can be compressed better
- and on the likelihood that they are used less frequently than the medium-sized fonts, the performance cost will be smaller.

Compressed fonts also support `bpp=3`.

Kerning

Fonts may provide kerning information to adjust the spacing between specific characters.

- The online converter generates kerning tables.
- The offline converter generates kerning tables unless `--no-kerning` is specified.
- FreeType integration does not currently support kerning.
- The Tiny TTF font engine supports GPOS (Glyph Positioning) and Kern tables.

To configure kerning at runtime, use `lv_font_set_kerning()`.

7.4.5 Adding a New Font

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

1. The simplest method is to use the Online font converter. Just set the parameters, click the *Convert* button, copy the font to your project and use it. **Be sure to carefully read the steps provided on that site or you will get an error while converting.**
2. Use the Offline font converter. (Requires Node.js to be installed)
3. If you want to create something like the built-in fonts (Montserrat font and symbols) but in a different size and/or ranges, you can use the `built_in_font_gen.py` script in `lvgl/scripts/built_in_font` folder. (This requires Python and https://github.com/lvgl/lv_font_conv/ to be installed.)

To declare a font in a file, use `LV_FONT_DECLARE(my_font_name)`.

To make fonts globally available (like the built-in fonts), add them to `LV_FONT_CUSTOM_DECLARE` in `lv_conf.h`.

7.4.6 Adding New Symbols

The built-in symbols are created from the FontAwesome font.

1. Search for a symbol on https://fontawesome.com. For example the USB symbol. Copy its Unicode ID which is `0xf287`.
2. Open the Online font converter. Add FontAwesome.woff.
3. Set the parameters such as Name, Size, BPP. You'll use this name to declare and use the font in your code.

4. Add the Unicode ID of the symbol to the range field. E.g.`0xf287` for the USB symbol. More symbols can be enumerated with `,`.
5. Convert the font and copy the generated source code to your project. Make sure to compile the `.c` file of your font.
6. Declare the font using `extern lv_font_t my_font_name;` or simply use `LV_FONT_DECLARE(my_font_name)`.

Using the symbol

1. Convert the Unicode value to UTF8, for example on this site. For `0xf287` the *Hex UTF-8 bytes* are `EF 8A 87`.
2. Create a `#define` string from the UTF8 values: `#define MY_USB_SYMBOL "\xEF\x8A\x87"`
3. Create a label and set the text. Eg. `lv_label_set_text(label, MY_USB_SYMBOL)`

note

`lv_label_set_text(label, MY_USB_SYMBOL)` searches for this symbol in the font defined in the style's `text.font` property. To use the symbol you will need to set the style's text font to use the generated font, e.g. `lv_style_set_text_font(&my_style, &my_font_name)` or `lv_obj_set_style_text_font(label, &my_font_name, 0)`.

7.4.7 Loading a Font at Run-Time

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

note

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

Esempio

```
lv_font_t *my_font = lv_binfont_create("X:/path/to/my_font.bin");
if(my_font == NULL) return;

/* Use the font */

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

7.4.8 Loading a Font from a Memory Buffer at Run-Time

`lv_binfont_create_from_buffer()` can be used to load a font from a memory buffer. This function may be useful to load a font from an external file system, which is not supported by LVGL. The font needs to be in the same format as if it were loaded from a file.

note

To load a font from a buffer *LVGL's filesystem* needs to be enabled and the MEMFS driver must be added.

Esempio

```
lv_font_t *my_font;
uint8_t *buf;
uint32_t bufsize;

/* Read font file into the buffer from the external file system */
...

/* Load font from the buffer */
```

(continues on next page)

(continua dalla pagina precedente)

```
my_font = lv_binfont_create_from_buffer((void *)buf, buf));
if(my_font == NULL) return;
/* Use the font */

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

7.4.9 Using a BDF Font

Small displays with low resolution don't look pretty with automatically rendered fonts. A bitmap font provides the solution, but it's necessary to convert the bitmap font (BDF) to a TTF.

Convert BDF to TTF

BDF are bitmap fonts where fonts are not described in outlines but in pixels. BDF files can be used but they must be converted into the TTF format using `mkttf`, which can be found in this GitHub repository: https://github.com/Tblue/mkttf . This tool uses potrace to generate outlines from the bitmap information. The bitmap itself will be embedded into the TTF as well. lv_font_conv uses the embedded bitmap but it also needs the outlines. One might think you can use a fake MS Bitmap only sfnt (ttf) (TTF without outlines) created by fontforge, but this will not work.

Install imagemagick, python3, python3-fontforge and potrace

On Ubuntu Systems, just type

```
sudo apt install imagemagick python3-fontforge potrace
```

Clone mkttf

```
git clone https://github.com/Tblue/mkttf
```

Read the mkttf docs.

Former versions of imagemagick needs the imagemagick call in front of convert, identify and so on. But newer versions don't. So you might want to change 2 lines in `potrace-wrapper.sh` --- open `potrace-wrapper.sh` and remove imagemagick from line 55 and line 64:

line 55

```
wh=($(identify -format '%[width]pt %[height]pt' "${input?}"))
```

line 64

```
convert "${input?}" -sample '1000%' - \
```

It might be necessary to change the mkttf.py script.

line 1

```
#!/usr/bin/env python3
```

Example for a 12px font

```
cd mkttf
./mkttf.py ./TerminusMedium-12-12.bdf
Importing bitmaps from 0 additional fonts...
Importing font `./TerminusMedium-12-12.bdf' into glyph background...
Processing glyphs...
Saving TTF file...
```

(continues on next page)

(continua dalla pagina precedente)

```
Saving SFD file...
Done!
```

The TTF `TerminusMedium-001.000.ttf` will be created from `./TerminusMedium-12-12.bdf`.

To create a font for LVGL:

```
lv_font_conv --bpp 1 --size 12 --no-compress --font TerminusMedium-001.000.
↪ttf --range 0x20-0x7e,0xa1-0xff --format lvgl -o terminus_1bpp_12px.c
```

note

use 1-bpp because we don't use anti-aliasing. It doesn't look sharp on displays with a low resolution.

7.4.10 Adding a New Font Engine

LVGL's font interface is designed to be very flexible but, even so, you can add your own font engine in place of LVGL's internal one. For example, you can use FreeType to real-time render glyphs from TTF fonts or use an external flash to store the font's bitmap and read them when the library needs them. FreeType can be used in LVGL as described in *Freetype*.

To add a new font engine, a custom `lv_font_t` variable needs to be created:

```
/* Describe the properties of a font */
lv_font_t my_font;
my_font.get_glyph_dsc = my_get_glyph_dsc_cb;        /* Set a callback to␣
↪get info about glyphs */
my_font.get_glyph_bitmap = my_get_glyph_bitmap_cb;  /* Set a callback to␣
↪get bitmap of a glyph */
my_font.line_height = height;                       /* The real line␣
↪height where any text fits */
my_font.base_line = base_line;                      /* Base line measured␣
↪from the top of line_height */
my_font.dsc = something_required;                   /* Store any␣
↪implementation specific data here */
my_font.user_data = user_data;                      /* Optionally some␣
↪extra user data */

...

/* Get info about glyph of `unicode_letter` in `font` font.
 * Store the result in `dsc_out`.
 * The next letter (`unicode_letter_next`) might be used to calculate the␣
↪width required by this glyph (kerning)
 */
bool my_get_glyph_dsc_cb(const lv_font_t * font, lv_font_glyph_dsc_t * dsc_
↪out, uint32_t unicode_letter, uint32_t unicode_letter_next)
{
    /* Your code here */

    /* Store the result.
     * For example ...
     */
    dsc_out->adv_w = 12;        /* Horizontal space required by the glyph␣
↪in [px] */
    dsc_out->box_h = 8;         /* Height of the bitmap in [px] */
    dsc_out->box_w = 6;         /* Width of the bitmap in [px] */
    dsc_out->ofs_x = 0;         /* X offset of the bitmap in [pf] */
```

(continues on next page)

(continua dalla pagina precedente)

```
    dsc_out->ofs_y = 3;         /* Y offset of the bitmap measured from␣
↪the as line */
    dsc_out->format= LV_FONT_GLYPH_FORMAT_A2;

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


/* Get the bitmap of `unicode_letter` from `font`. */
const uint8_t * my_get_glyph_bitmap_cb(const lv_font_t * font, uint32_t␣
↪unicode_letter)
{
    /* Your code here */

    /* The bitmap should be a continuous bitstream where
     * each pixel is represented by `bpp` bits */

    return bitmap;    /* Or NULL if not found */
}
```

7.4.11 Using Font Fallback

If the font in use does not have a glyph needed in a text-rendering task, you can specify a `fallback` font to be used in `lv_font_t`.

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

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

7.4.12 API

lv_font.h

lv_font_fmt_txt.h

lv_font_fmt_txt_private.h

lv_font_manager.h

lv_font_manager_recycle.h

7.5 Images (lv_image)

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

7.5.1 Store images

You can store images in two places

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

Variables

Images stored internally in a variable are composed mainly of an `lv_image_dsc_t` structure with the following fields:

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

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

Files

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

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

7.5.2 Color formats

Various built-in color formats are supported:

- `LV_COLOR_FORMAT_NATIVE`: Simply stores the RGB colors (in whatever color depth LVGL is configured for).
- `LV_COLOR_FORMAT_NATIVE_WITH_ALPHA`: Like `LV_COLOR_FORMAT_NATIVE` but it also adds an alpha (transparency) byte for every pixel.
- `LV_COLOR_FORMAT_I1`, `LV_COLOR_FORMAT_I2`, `LV_COLOR_FORMAT_I4`, `LV_COLOR_FORMAT_I8`: Uses a palette with 2, 4, 16 or 256 colors and stores each pixel in 1, 2, 4 or 8 bits.
- `LV_COLOR_FORMAT_A1`, `LV_COLOR_FORMAT_A2`, `LV_COLOR_FORMAT_A4`, `LV_COLOR_FORMAT_A8`: **Only stores the Alpha value with 1, 2, 4 or 8 bits.** The pixels take the color of `style.img_recolor` and the set opacity. The source image has to be an alpha channel. This is ideal for bitmaps similar to fonts where the whole image is one color that can be altered.

The bytes of `LV_COLOR_FORMAT_NATIVE` images are stored in the following order.

- 32-bit color depth:
  - **Byte 0**: Blue
  - **Byte 1**: Green

  - **Byte 2**: Red
  - **Byte 3**: Alpha (only with `LV_COLOR_FORMAT_NATIVE_WITH_ALPHA`)
- 16-bit color depth:
  - **Byte 0**: Green 3 lower bit, Blue 5 bit
  - **Byte 1**: Red 5 bit, Green 3 higher bit
  - **Byte 2**: Alpha byte (only with `LV_COLOR_FORMAT_NATIVE_WITH_ALPHA`)
- 8-bit color depth:
  - **Byte 0**: Red 3 bit, Green 3 bit, Blue 2 bit
  - **Byte 2**: Alpha byte (only with `LV_COLOR_FORMAT_NATIVE_WITH_ALPHA`)

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

- `LV_COLOR_FORMAT_RAW`: Indicates a basic raw image (e.g. a PNG or JPG image).
- `LV_COLOR_FORMAT_RAW_ALPHA`: Indicates that an image has alpha and an alpha byte is added for every pixel.

7.5.3 Add and use images

You can add images to LVGL in two ways:

- using the online converter
- manually create images

Online converter

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

Adding an image to LVGL via the online converter is easy.

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

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

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

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

Manually create an image

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

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

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

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

Use images

The simplest way to use an image in LVGL is to display it with an *Image (lv_image)* Widget:

```
lv_obj_t * icon = lv_image_create(lv_screen_active(), NULL);

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

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

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

7.5.4 Image decoder

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

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

An image decoder consists of 4 callbacks:

info
: get some basic info about the image (width, height and color format).

open
: open an image: - store a decoded image - set it to `NULL` to indicate the image can be read line-by-line.

get_area
: if *open* didn't fully open an image this function should give back part of image as decoded data.

close
: close an opened image, free the allocated resources.

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

The following formats are understood by the built-in decoder:

- *LV_COLOR_FORMAT_I1*
- *LV_COLOR_FORMAT_I2*

- *LV_COLOR_FORMAT_I4*
- *LV_COLOR_FORMAT_I8*
- *LV_COLOR_FORMAT_RGB888*
- *LV_COLOR_FORMAT_XRGB8888*
- *LV_COLOR_FORMAT_ARGB8888*
- *LV_COLOR_FORMAT_RGB565*
- *LV_COLOR_FORMAT_RGB565A8*

Custom image formats

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

`header.cf` will be *LV_COLOR_FORMAT_RAW*, *LV_COLOR_FORMAT_RAW_ALPHA* accordingly. You should choose the correct format according to your needs: a fully opaque image, using an alpha channel.

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

Registering an image decoder

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

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

```
/* Create a new decoder and register functions */
lv_image_decoder_t * dec = lv_image_decoder_create();
lv_image_decoder_set_info_cb(dec, decoder_info);
lv_image_decoder_set_open_cb(dec, decoder_open);
lv_image_decoder_set_get_area_cb(dec, decoder_get_area);
lv_image_decoder_set_close_cb(dec, decoder_close);


/**
 * Get info about a PNG image
 * @param decoder   pointer to the decoder where this function belongs
 * @param src       can be file name or pointer to a C array
 * @param header    image information is set in header parameter
 * @return          LV_RESULT_OK: no error; LV_RESULT_INVALID: can't get␣
 ↪the info
 */
static lv_result_t decoder_info(lv_image_decoder_t * decoder, const void *␣
 ↪src, lv_image_header_t * header)
{
  /* Check whether the type `src` is known by the decoder */
  if(is_png(src) == false) return LV_RESULT_INVALID;

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

  header->cf = LV_COLOR_FORMAT_ARGB8888;
  header->w = width;
  header->h = height;
}
```

(continues on next page)

(continua dalla pagina precedente)

```
/**
 * Open a PNG image and decode it into dsc.decoded
 * @param decoder   pointer to the decoder where this function belongs
 * @param dsc       image descriptor
 * @return          LV_RESULT_OK: no error; LV_RESULT_INVALID: can't open␣
↪the image
 */
static lv_result_t decoder_open(lv_image_decoder_t * decoder, lv_image_
↪decoder_dsc_t * dsc)
{
  (void) decoder; /* Unused */

  /* Check whether the type `src` is known by the decoder */
  if(is_png(dsc->src) == false) return LV_RESULT_INVALID;

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

  /* Change the color format if decoded image format is different than␣
↪original format. For PNG it's usually decoded to ARGB8888 format */
  dsc->decoded.header.cf = LV_COLOR_FORMAT_...

  /* Call a binary image decoder function if required. It's not required␣
↪if `my_png_decoder` opened the image in true color format. */
  lv_result_t res = lv_bin_decoder_open(decoder, dsc);

  return res;
}

/**
 * Decode an area of image
 * @param decoder      pointer to the decoder where this function belongs
 * @param dsc          image decoder descriptor
 * @param full_area    input parameter. the full area to decode after␣
↪enough subsequent calls
 * @param decoded_area input+output parameter. set the values to `LV_COORD_
↪MIN` for the first call and to reset decoding.
 *                     the decoded area is stored here after each call.
 * @return             LV_RESULT_OK: ok; LV_RESULT_INVALID: failed or␣
↪there is nothing left to decode
 */
static lv_result_t decoder_get_area(lv_image_decoder_t * decoder, lv_image_
↪decoder_dsc_t * dsc,
                                    const lv_area_t * full_area, lv_area_t *␣
↪decoded_area)
{
  /**
  * If `dsc->decoded` is always set in `decoder_open` then `decoder_get_
↪area` does not need to be implemented.
  * If `dsc->decoded` is only sometimes set or never set in `decoder_open`␣
↪then `decoder_get_area` is used to
  * incrementally decode the image by calling it repeatedly until it␣
↪returns `LV_RESULT_INVALID`.
  * In the example below the image is decoded line-by-line but the decoded␣
```

(continues on next page)

(continua dalla pagina precedente)

```
↪area can have any shape and size
 * depending on the requirements and capabilities of the image decoder.
 */

 my_decoder_data_t * my_decoder_data = dsc->user_data;

 /* if `decoded_area` has a field set to `LV_COORD_MIN` then reset␣
↪decoding */
 if(decoded_area->y1 == LV_COORD_MIN) {
   decoded_area->x1 = full_area->x1;
   decoded_area->x2 = full_area->x2;
   decoded_area->y1 = full_area->y1;
   decoded_area->y2 = decoded_area->y1; /* decode line-by-line, starting␣
↪with the first line */

   /* create a draw buf the size of one line */
   bool reshape_success = NULL != lv_draw_buf_reshape(my_decoder_data->
↪partial,
                                                      dsc->decoded.header.
↪cf,
                                                      lv_area_get_
↪width(full_area),
                                                      1,
                                                      LV_STRIDE_AUTO);
   if(!reshape_success) {
     lv_draw_buf_destroy(my_decoder_data->partial);
     my_decoder_data->partial = lv_draw_buf_create(lv_area_get_width(full_
↪area),
                                                   1,
                                                   dsc->decoded.header.cf,
                                                   LV_STRIDE_AUTO);

     my_png_decode_line_reset(full_area);
   }
 }
 /* otherwise decoding is already in progress. decode the next line */
 else {
   /* all lines have already been decoded. indicate completion by␣
↪returning `LV_RESULT_INVALID` */
   if (decoded_area->y1 >= full_area->y2) return LV_RESULT_INVALID;
   decoded_area->y1++;
   decoded_area->y2++;
 }

 my_png_decode_line(my_decoder_data->partial);

 return LV_RESULT_OK;
}

/**
 * Close PNG image and free data
 * @param decoder   pointer to the decoder where this function belongs
 * @param dsc       image decoder descriptor
 * @return          LV_RESULT_OK: no error; LV_RESULT_INVALID: can't open␣
↪the image
 */
```

(continues on next page)

(continua dalla pagina precedente)

```
static void decoder_close(lv_image_decoder_t * decoder, lv_image_decoder_
↪dsc_t * dsc)
{
  /* Free all allocated data */
  my_png_cleanup();

  my_decoder_data_t * my_decoder_data = dsc->user_data;
  lv_draw_buf_destroy(my_decoder_data->partial);

  /* Call the built-in close function if the built-in open/get_area was␣
↪used */
  lv_bin_decoder_close(decoder, dsc);

}
```

So in summary:

- In `decoder_info`, you should collect some basic information about the image and store it in `header`.
- In `decoder_open`, you should try to open the image source pointed by `dsc->src`. Its type is already in `dsc->src_type == LV_IMG_SRC_FILE/VARIABLE`. If this format/type is not supported by the decoder, return `LV_RESULT_INVALID`. However, if you can open the image, a pointer to the decoded image should be set in `dsc->decoded`. If the format is known, but you don't want to decode the entire image (e.g. no memory for it), set `dsc->decoded = NULL` and use `decoder_get_area` to get the image area pixels.
- In `decoder_close` you should free all allocated resources.
- `decoder_get_area` is optional. In this case you should decode the whole image In `decoder_open` function and store image data in `dsc->decoded`. Decoding the whole image requires extra memory and some computational overhead.

Manually use an image decoder

LVGL will use registered image decoders automatically if you try and draw a raw image (i.e. using the `lv_image` Widget) but you can use them manually as well. Create an `lv_image_decoder_dsc_t` variable to describe the decoding session and call *`lv_image_decoder_open()`*.

The `color` parameter is used only with `LV_COLOR_FORMAT_A1/2/4/8` images to tell color of the image.

```
lv_result_t res;
lv_image_decoder_dsc_t dsc;
lv_image_decoder_args_t args = { 0 }; /* Custom decoder behavior via args␣
↪*/
res = lv_image_decoder_open(&dsc, &my_img_dsc, &args);

if(res == LV_RESULT_OK) {
  /* Do something with `dsc->decoded`. You can copy out the decoded image␣
↪by `lv_draw_buf_dup(dsc.decoded)`*/
  lv_image_decoder_close(&dsc);
}
```

Image post-processing

Considering that some hardware has special requirements for image formats, such as alpha premultiplication and stride alignment, most image decoders (such as PNG decoders) may not directly output image data that meets hardware requirements.

For this reason, LVGL provides a solution for image post-processing. First, call a custom post-processing function after `lv_image_decoder_open` to adjust the data in the image cache, and then mark the processing status in `cache_entry->process_state` (to avoid repeated post-processing).

See the detailed code below:

- Stride alignment and premultiply post-processing example:

```
/* Define post-processing state */
typedef enum {
  IMAGE_PROCESS_STATE_NONE = 0,
  IMAGE_PROCESS_STATE_STRIDE_ALIGNED = 1 << 0,
  IMAGE_PROCESS_STATE_PREMULTIPLIED_ALPHA = 1 << 1,
} image_process_state_t;

lv_result_t my_image_post_process(lv_image_decoder_dsc_t * dsc)
{
  lv_color_format_t color_format = dsc->header.cf;
  lv_result_t res = LV_RESULT_OK;

  if(color_format == LV_COLOR_FORMAT_ARGB8888) {
    lv_cache_lock();
    lv_cache_entry_t * entry = dsc->cache_entry;

    if(!(entry->process_state & IMAGE_PROCESS_STATE_PREMULTIPLIED_ALPHA)) {
      lv_draw_buf_premultiply(dsc->decoded);
      LV_LOG_USER("premultiplied alpha OK");

      entry->process_state |= IMAGE_PROCESS_STATE_PREMULTIPLIED_ALPHA;
    }

    if(!(entry->process_state & IMAGE_PROCESS_STATE_STRIDE_ALIGNED)) {
       uint32_t stride_expect = lv_draw_buf_width_to_stride(decoded->
↪header.w, decoded->header.cf);
       if(decoded->header.stride != stride_expect) {
           LV_LOG_WARN("Stride mismatch");
           lv_draw_buf_t * aligned = lv_draw_buf_adjust_stride(decoded,␣
↪stride_expect);
           if(aligned == NULL) {
               LV_LOG_ERROR("No memory for Stride adjust.");
               return NULL;
           }

           decoded = aligned;
       }

       entry->process_state |= IMAGE_PROCESS_STATE_STRIDE_ALIGNED;
    }

alloc_failed:
    lv_cache_unlock();
  }

  return res;
}
```

- GPU draw unit example:

```
void gpu_draw_image(lv_draw_unit_t * draw_unit, const lv_draw_image_dsc_t␣
↪* draw_dsc, const lv_area_t * coords)
{
  ...
  lv_image_decoder_dsc_t decoder_dsc;
  lv_result_t res = lv_image_decoder_open(&decoder_dsc, draw_dsc->src,␣
↪NULL);
  if(res != LV_RESULT_OK) {
    LV_LOG_ERROR("Failed to open image");
    return;
  }

  res = my_image_post_process(&decoder_dsc);
  if(res != LV_RESULT_OK) {
    LV_LOG_ERROR("Failed to post-process image");
    return;
  }
  ...
}
```

7.5.5 Image caching

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

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

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

Cache size

The size of cache (in bytes) can be defined with *`LV_CACHE_DEF_SIZE`* in *lv_conf.h*. The default value is 0, so no image is cached.

The size of cache can be changed at run-time with *`lv_cache_set_max_size`*`(size_t size)`, and get with *`lv_cache_get_max_size`*`()`.

Value of images

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

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

If you want or need to override LVGL's measurement, you can manually set the *weight* value in the cache entry in `cache_entry->weight = time_ms` to give a higher or lower value. (Leave it unchanged to let LVGL control it.)

Every cache entry has a *"life"* value. Every time an image is opened through the cache, the *life* value of all entries is increased by their *weight* values to make them older. When a cached image is used, its *usage_count* value is increased to make it more alive.

If there is no more space in the cache, the entry with *usage_count == 0* and lowest life value will be dropped.

Memory usage

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

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

Clean the cache

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

To do this, use `lv_cache_invalidate(lv_cache_find(&my_png, LV_CACHE_SRC_TYPE_PTR, 0, 0))`.

Custom cache algorithm

If you want to implement your own cache algorithm, you can refer to the following code to replace the LVGL built-in cache manager:

```
static lv_cache_entry_t * my_cache_add_cb(size_t size)
{
  ...
}

static lv_cache_entry_t * my_cache_find_cb(const void * src, lv_cache_src_
↪type_t src_type, uint32_t param1, uint32_t param2)
{
  ...
}

static void my_cache_invalidate_cb(lv_cache_entry_t * entry)
{
  ...
}

static const void * my_cache_get_data_cb(lv_cache_entry_t * entry)
{
  ...
}

static void my_cache_release_cb(lv_cache_entry_t * entry)
{
  ...
}

static void my_cache_set_max_size_cb(size_t new_size)
{
  ...
}

static void my_cache_empty_cb(void)
{
  ...
}
```

(continues on next page)

(continua dalla pagina precedente)

```
void my_cache_init(void)
{
 /* Initialize new cache manager. */
 lv_cache_manager_t my_manager;
 my_manager.add_cb = my_cache_add_cb;
 my_manager.find_cb = my_cache_find_cb;
 my_manager.invalidate_cb = my_cache_invalidate_cb;
 my_manager.get_data_cb = my_cache_get_data_cb;
 my_manager.release_cb = my_cache_release_cb;
 my_manager.set_max_size_cb = my_cache_set_max_size_cb;
 my_manager.empty_cb = my_cache_empty_cb;

 /* Replace existing cache manager with the new one. */
 lv_cache_lock();
 lv_cache_set_manager(&my_manager);
 lv_cache_unlock();
}
```

7.5.6 API

lv_api_map_v8.h

lv_api_map_v9_0.h

lv_api_map_v9_1.h

lv_draw_buf.h

lv_draw_image.h

lv_draw_image_private.h

lv_image.h

lv_image_cache.h

lv_image_decoder.h

lv_image_decoder_private.h

lv_image_dsc.h

lv_image_header_cache.h

lv_image_private.h

lv_obj_property_names.h

7.6 Timer (lv_timer)

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

By default, LVGL itself uses Timers to:

- refresh each display --- during the creation of each *Display (lv_display)*, a Timer is created for that Display. That Timer refreshes the display based on the configured value of `LV_DEF_REFR_PERIOD`, and also sends all display-related events, like `LV_EVENT_REFR_START`, `LV_EVENT_REFR_READY`, etc.
- read input devices --- during the creation of each *Input Device (lv_indev)*, a Timer is created for that Input Device based on the configured value of `LV_DEF_REFR_PERIOD`. That Timer causes that input device to be read and also sends all input-device-related events, like `LV_EVENT_CLICKED`, `LV_EVENT_PRESSED`, etc.

- update system-monitor values --- if *LV_USE_SYSMON* is set to `1` in `lv_conf.h`, one or more timers are created to periodically compute and monitor system performance statistics and LVGL's memory usage.

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

7.6.1 Creating a Timer

To create a new Timer, use *lv_timer_create*`(timer_cb, period_ms, user_data)`. It returns an `lv_timer_t *` which can be used later to modify the parameters of the Timer, pause it, or delete it when it is no longer needed. *lv_timer_create_basic()* can also be used to create a new Timer without specifying any parameters.

A Timer callback should have this prototype: `void (*lv_timer_cb_t)(lv_timer_t *)`.

Ad esempio:

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

  /* Do something with LVGL */
  if(something_happened) {
    something_happened = false;
    lv_button_create(lv_screen_active());
  }
}

...

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

7.6.2 Ready and Reset

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

lv_timer_reset`(timer)` resets the period of a Timer. It will be called again after its currently-set period (in milliseconds) has elapsed.

See what happens when *no timers are ready* and *lv_timer_handler()* is called.

7.6.3 Setting Parameters

You can modify these Timer parameters at any time during its life:

- *lv_timer_set_cb*`(timer, new_cb)`
- *lv_timer_set_period*`(timer, new_period_ms)`
- *lv_timer_set_user_data*`(timer, user_data)`

7.6.4 Repeat Count

When a Timer is created, its repeat-count is set to `-1` to cause it to repeat indefinitely. You can make a Timer repeat only a given number of times with *lv_timer_set_repeat_count*`(timer, count)`. By default, once the Timer has run `count` times, it will be automatically deleted.

You can use *lv_timer_set_auto_delete*`(timer, false)` if you want the timer to instead be paused after it has run `count` times. This can be handy if you reuse that timer repeatedly and want to avoid the CPU and *lv_malloc()* overhead of repeatedly creating and deleting a timer. If you use this option, you will need to set its

repeat count (to either `-1` or a positive repeat count, since it will have decremented to `0`) and *resume* it to make it active again.

7.6.5 Pause and Resume

`lv_timer_pause(timer)` pauses the specified Timer.

`lv_timer_resume(timer)` resumes the specified Timer.

7.6.6 Measuring Idle Time

You can get the idle percentage time of `lv_timer_handler()` with `lv_timer_get_idle()`. Note that it does not measure the idle time of the overall system, only of `lv_timer_handler()`. This can be misleading if you are using an operating system and DMA and/or GPU are used during rendering, as it does not actually measure the time the OS spends in an idle thread.

If you are using an OS and wish to get the time the CPU is spending in an idle thread, one way of doing so is configuring `LV_USE_SYSMON` and `LV_USE_PERF_MONITOR` to `1` in `lv_conf.h` (if they are not already), and setting the macro `LV_SYSMON_GET_IDLE` to the name of a function that fetches the percent of CPU time spent in the OS's idle thread. An example of such a function is `lv_os_get_idle_percent()` in `lv_freertos.c`. While the configuration is set this way, some system performance statistics (including CPU load) will appear on the display in a partially-transparent label whose location is set by the `LV_USE_PERF_MONITOR_POS` macro.

7.6.7 Enable and Disable

You can temporarily disable Timer handling with `lv_timer_enable(false)`. Be advised: this also pauses handling of Timers that refresh Display(s) and read from input devices, so don't forget to re-enable it with `lv_timer_enable(true)` as soon as the need for the pause is over.

7.6.8 Timer Handler Resume Callback

When the Timer system has been disabled (causing `lv_timer_handler()` to return early before it has processed any timers), if you want to take some action when the Timer system is re-enabled again, set a resume callback using `lv_timer_handler_set_resume_cb(cb, user_data)`. The callback should have this prototype: `void (*lv_timer_handler_resume_cb_t)(void*)`.

7.6.9 Asynchronous calls

There are several cases in which you may not want to perform an action immediately. Some examples are:

- you cannot delete a Widget because something else is still using it,
- you don't want to block execution now, or
- you detect the need to delete a Widget in a thread other than the thread making LVGL calls (e.g. in a case where you are using a *Gateway Thread* to make all LVGL calls in a multi-threaded environment).

For these cases, `lv_async_call(my_function, data_p)` can be used to call `my_function` on the next invocation of `lv_timer_handler()`. As a side effect, this also ensures it is called in a thread in which it is safe to make LVGL calls. `data_p` will be passed to the function when it's called. Note that only the data's pointer is saved, so whatever it is pointing to needs to remain valid until the function is called, so it can point to `static`, global or dynamically allocated data. If you want to cancel an asynchronous call, call `lv_async_call_cancel(my_function, data_p)`, which will remove all asynchronous calls matching `my_function` and `data_p`.

Note that if `lv_async_call(my_function, data_p)` is called from a thread other than the one that normally makes LVGL calls, you are still obligated to protect the LVGL data structure using a MUTEX.

Ad esempio:

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

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

...

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

/* Delete screen on next call of `lv_timer_handler`, not right now. */
lv_lock();
lv_async_call(my_screen_clean_up, lv_screen_active());
lv_unlock();

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

If you just want to delete a Widget and don't need to clean anything up in `my_screen_cleanup` you could just use *`lv_obj_delete_async`*`(widget)` which will delete the Widget on the next call to *`lv_timer_handler()`*.

7.6.10 API

lv_api_map_v8.h

lv_timer.h

lv_timer_private.h

7.7 Animation (lv_anim)

Animations allow you to define the way something should move or change over time, and let LVGL do the heavy lifting of making it happen. What makes it so powerful is that the thing being changed can be virtually anything in your system. It is very convenient to apply this to LVGL Widgets in your user interface (UI), to change their appearance, size or location over time. But because it is --- at its core --- a generic change-over-time manager, complete with a variety of optional event callbacks, its application can be wider than just to UI components.

For each Animation you create, it accomplishes the above by providing a generic method of varying a signed integer from a start value to an end value over a specified time period. It allows you to specify what object it applies to (the "variable"), which is available in the callback functions that are called as the Animation is playing through.

This variation over time can be linear (default), it can be on a path (curve) that you specify, and there is even a variety of commonly-used non-linear effects that can be specified.

The main callback called during an Animation (when it is playing) is called an *animator* function, which has the following prototype:

```
void func(void *var , int32_t value);
```

This prototype makes it easy to use most of the LVGL *set* functions directly or via a trivial wrapper. It includes:

- most of the widget properties
- functions that set *local style properties* directly on objects (needs a wrapper to set the *selector*)
- set properties on *`lv_style_t`* objects (e.g. *shared styles*) (`lv_obj_report_style_change` needs to be called to notify the widgets having the style)
- `lv_style_set_<property_name>(&style, <value>)`

- `lv_obj_set_<property_name>(widget, <value>)`

Because of the former, an animation on a single `lv_style_t` object shared among several objects can simultaneously modify the appearance of all objects that use it. See *Stili* for more details.

Examples of the latter are: `lv_obj_set_x(widget, value)` or `lv_obj_set_width(widget, value)`.

This makes it very convenient to apply to the appearance (and other attributes) of UI components. But you can provide your own "set" functions, and so the application of Animations is really limited only by your imagination.

The number of Animations that can be playing at the same time for a given object with a given *animator* callback is one (1). However, the number of Animations that can be playing at the same time is limited only by available RAM and CPU time for:

- a given object with different *animator* callbacks; and
- different objects.

Thus, you can have a Button's width being changed by one Animation while having its height being changed by another Animation.

7.7.1 Create an Animation

To create an Animation, start by creating an Animation *template* in an `lv_anim_t` variable. It has to be initialized and configured with `lv_anim_set_...()` functions.

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

static lv_anim_t   anim_template;
static lv_anim_t * running_anim;

lv_anim_init(&anim_template);

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

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

/* Set target of the Animation */
lv_anim_set_var(&anim_template, widget);

/* Length of the Animation [ms] */
lv_anim_set_duration(&anim_template, duration_in_ms);

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

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

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

/* Set path (curve). Default is linear */
lv_anim_set_path_cb(&anim_template, lv_anim_path_ease_in);

/* Set anim_template callback to indicate when the Animation is completed.␣
↪*/
```

(continues on next page)

(continua dalla pagina precedente)

```
lv_anim_set_completed_cb(&anim_template, completed_cb);

/* Set anim_template callback to indicate when the Animation is deleted␣
↪(idle). */
lv_anim_set_deleted_cb(&anim_template, deleted_cb);

/* Set anim_template callback to indicate when the Animation is started␣
↪(after delay). */
lv_anim_set_start_cb(&anim_template, start_cb);

/* When ready, play the Animation backward with this duration. Default is␣
↪0 (disabled) [ms] */
lv_anim_set_reverse_duration(&anim_template, time);

/* Delay before reverse play. Default is 0 (disabled) [ms] */
lv_anim_set_reverse_delay(&anim_template, delay);

/* Number of repetitions. Default is 1. LV_ANIM_REPEAT_INFINITE for␣
↪infinite repetition */
lv_anim_set_repeat_count(&anim_template, cnt);

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

/* true (default): apply the start value immediately, false: apply start␣
↪value after delay when the Anim. really starts. */
lv_anim_set_early_apply(&anim_template, true/false);

/* START THE ANIMATION
 *------------------*/
running_anim = lv_anim_start(&anim_template);   /* Start the Animation */
```

7.7.2 Animation Path

You can control the Path (curve) of an Animation. The simplest case is linear, meaning the current value between *start* and *end* is changed at the same rate (i.e. with fixed steps) over the duration of the Animation. A *Path* is a function which calculates the next value to set based on the current state of the Animation. There are a number of built-in *Paths* that can be used:

- `lv_anim_path_linear()`: linear Animation (default)
- `lv_anim_path_step()`: change in one step at the end
- `lv_anim_path_ease_in()`: slow at the beginning
- `lv_anim_path_ease_out()`: slow at the end
- `lv_anim_path_ease_in_out()`: slow at the beginning and end
- `lv_anim_path_overshoot()`: overshoot the end value
- `lv_anim_path_bounce()`: bounce back a little from the end value (like hitting a wall)

Alternately, you can provide your own Path function.

`lv_anim_init(&my_anim)` sets the Path to `lv_anim_path_linear()` by default. If you want to use a different Path (including a custom Path function you provide), you set it using `lv_anim_set_path_cb(&anim_template, path_cb)`.

If you provide your own custom Path function, its prototype is:

```
int32_t   calculate_value(lv_anim_t * anim);
```

7.7.3 Speed vs Time

Normally, you set the Animation duration directly using `lv_anim_set_duration(&anim_template, duration_in_ms)`. But in some cases the *rate* is known but the duration is not known. Given an Animation's `start` and `end` values, *rate* here means the number of units of change per second, i.e. how quickly (units per second) the Animation's value needs to change between the `start` and `end` value. For such cases there is a utility function `lv_anim_speed_to_time()` you can use to compute the Animation's duration, so you can set it like this:

```
uint32_t  change_per_sec = 20;
uint32_t  duration_in_ms = lv_anim_speed_to_time(change_per_sec, 0, 100);
/* `duration_in_ms` will be 5000 */
lv_anim_set_duration(&anim_template, duration_in_ms);
```

7.7.4 Animating in Both Directions

Sometimes an Animation needs to play forward, and then play backwards, effectively reversing course, animating from the `end` value back to the `start` value again. To do this, pass a non-zero value to this function to set the duration for the reverse portion of the Animation: `lv_anim_set_reverse_duration(&anim_template, duration_in_ms)`.

Optionally, you can also introduce a delay between the forward and backward directions using `lv_anim_set_reverse_delay(&anim_template, delay_in_ms)`

7.7.5 Starting an Animation

After you have set up your `lv_anim_t` object, it is important to realize that what you have set up is a "template" for a live, running Animation that has not been created yet. When you call `lv_anim_start(&anim_template)` passing the *template* you have set up, it uses your template to dynamically allocate an internal object that is a *live, running* Animation. This function returns a pointer to that object.

```
static lv_anim_t   anim_template;
static lv_anim_t * running_anim;

/* Set up template... */
lv_anim_init(&anim_template);
/* ...and other set-up functions above. */

/* Later... */
running_anim = lv_anim_start(&anim_template);
```

> **Nota**
>
> `lv_anim_start(&anim_template)` makes its own copy of the Animation template, so if you do not need it later, its contents do not need to be preserved after this call.

Once a *live running* Animation has been started, it runs until it has completed, or until it is deleted (see below), whichever comes first. An Animation has completed when:

- its "value" has reached the designated `end` value;
- if the Animation has a non-zero *reverse* duration value, then its value has run from the `end` value back to the `start` value again;
- if a non-zero repeat count has been set, it has repeated the Animation that number of times.

Once the *live, running* Animation reaches completion, it is automatically deleted from the list of running Animations. This does not impact your Animation template.

> **Nota**
>
> If `lv_anim_set_repeat_count(&anim_template, cnt)` has been called passing `LV_ANIM_REPEAT_INFINITE`, the animation never reaches a state of being "completed". In this case, it must be deleted to terminate the Animation.

7.7.6 Deleting Animations

You should delete an Animation using `lv_anim_delete(var, func)` if one of these two conditions exists:

- the object (variable) being animated is deleted (and it is not a Widget) or
- a running animation needs to be stopped before it is completed.

> **Nota**
>
> If the object (variable) being deleted is a type of Widget, the housekeeping code involved in deleting it also deletes any running animations that are connected with it. So `lv_anim_delete(var, func)` only needs to be called if the object being deleted is *not* one of the Widgets.

If you kept a copy of the pointer returned by `lv_anim_start()` as `running_anim`, you can delete the running animation like this:

```
lv_anim_delete(running_anim->var, running_anim->exec_cb);
```

In the event that the Animation completes *after* you have determined it needs to be deleted, and before the call to `lv_anim_delete()` is made, it does no harm to call it a second time --- no damage will occur.

This function returns a Boolean value indicating whether any *live, running* Animations were deleted.

7.7.7 Pausing Animations

If you kept a copy of the pointer returned by `lv_anim_start()`, you can pause the running animation using `lv_anim_pause(animation)` and then resume it using `lv_anim_resume(animation)`.

`lv_anim_pause_for(animation, milliseconds)` is also available if you wish for the animation to resume automatically after.

7.7.8 Timeline

You can create a series of related animations that are linked together using an Animation Timeline. A Timeline is a collection of multiple Animations which makes it easy to create complex composite Animations. To create and use an Animation Timeline:

- Create an Animation template but do not call `lv_anim_start()` on it.
- Create an Animation Timeline object by calling `lv_anim_timeline_create()`.
- Add Animation templates to the Timeline by calling `lv_anim_timeline_add(timeline, start_time, &anim_template)`. `start_time` is the start time of the Animation on the Timeline. Note that `start_time` will override any value given to `lv_anim_set_delay(&anim_template, delay)`.
- Call `lv_anim_timeline_start(timeline)` to start the Animation Timeline.

Nota

`lv_anim_timeline_add(timeline, start_time, &anim_template)` makes its own copy of the contents of the Animation template, so if you do not need it later, its contents do not need to be preserved after this call.

It supports forward and reverse play of the entire Animation group, using `lv_anim_timeline_set_reverse(timeline, reverse)`. Note that if you want to play in reverse from the end of the Timeline, you need to call `lv_anim_timeline_set_progress(timeline, LV_ANIM_TIMELINE_PROGRESS_MAX)` after adding all Animations and before telling it to start playing.

Call `lv_anim_timeline_pause(timeline)` to pause the Animation Timeline. Note: this does not preserve its state. The only way to start it again is to call `lv_anim_timeline_start(timeline)`, which starts the Timeline from the beginning or at the point set by `lv_anim_timeline_set_progress(timeline, progress)`.

Call `lv_anim_timeline_set_progress(timeline, progress)` function to set the state of the Animation Timeline according to the `progress` value. `progress` is a value between `0` and `32767` (`LV_ANIM_TIMELINE_PROGRESS_MAX`) to indicate the proportion of the Timeline that has "played". Example: a `progress` value of `LV_ANIM_TIMELINE_PROGRESS_MAX / 2` would set the Timeline play to its half-way point.

Call `lv_anim_timeline_get_playtime(timeline)` function to get the total duration (in milliseconds) of the entire Animation Timeline.

Call `lv_anim_timeline_get_reverse(timeline)` function to get whether the Animation Timeline is also played in reverse after its forward play completes.

Call `lv_anim_timeline_delete(timeline)` function to delete the Animation Timeline. **Note**: If you need to delete a Widget during Animation, be sure to delete the Animation Timeline before deleting the Widget. Otherwise, the program may crash or behave abnormally.

7.7.9 Esempi

Avviare animazione su un evento

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_SWITCH

static void anim_x_cb(void * var, int32_t v)
{
    lv_obj_set_x((lv_obj_t *) var, v);
}

static void sw_event_cb(lv_event_t * e)
{
    lv_obj_t * sw = lv_event_get_target_obj(e);
    lv_obj_t * label = (lv_obj_t *) lv_event_get_user_data(e);

    if(lv_obj_has_state(sw, LV_STATE_CHECKED)) {
        lv_anim_t a;
        lv_anim_init(&a);
        lv_anim_set_var(&a, label);
        lv_anim_set_values(&a, lv_obj_get_x(label), 100);
        lv_anim_set_duration(&a, 500);
        lv_anim_set_exec_cb(&a, anim_x_cb);
        lv_anim_set_path_cb(&a, lv_anim_path_overshoot);
        lv_anim_start(&a);
    }
    else {
        lv_anim_t a;
        lv_anim_init(&a);
        lv_anim_set_var(&a, label);
        lv_anim_set_values(&a, lv_obj_get_x(label), -lv_obj_get_
↪width(label));
        lv_anim_set_duration(&a, 500);
        lv_anim_set_exec_cb(&a, anim_x_cb);
        lv_anim_set_path_cb(&a, lv_anim_path_ease_in);
        lv_anim_start(&a);
    }

}

/**
 * Start animation on an event
 */
void lv_example_anim_1(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Hello animations!");
    lv_obj_set_pos(label, 100, 10);

    lv_obj_t * sw = lv_switch_create(lv_screen_active());
    lv_obj_center(sw);
    lv_obj_add_state(sw, LV_STATE_CHECKED);
    lv_obj_add_event_cb(sw, sw_event_cb, LV_EVENT_VALUE_CHANGED, label);
}

#endif
```

Riproduzione di animazione

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_SWITCH

static void anim_x_cb(void * var, int32_t v)
{
    lv_obj_set_x((lv_obj_t *) var, v);
}

static void anim_size_cb(void * var, int32_t v)
{
    lv_obj_set_size((lv_obj_t *) var, v, v);
}

/**
 * Create a playback animation
 */
void lv_example_anim_2(void)
{

    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_remove_flag(obj, LV_OBJ_FLAG_SCROLLABLE);
    lv_obj_set_style_bg_color(obj, lv_palette_main(LV_PALETTE_RED), 0);
    lv_obj_set_style_radius(obj, LV_RADIUS_CIRCLE, 0);

    lv_obj_align(obj, LV_ALIGN_LEFT_MID, 10, 0);

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_var(&a, obj);
    lv_anim_set_values(&a, 10, 50);
    lv_anim_set_duration(&a, 1000);
    lv_anim_set_reverse_delay(&a, 100);
    lv_anim_set_reverse_duration(&a, 300);
    lv_anim_set_repeat_delay(&a, 500);
    lv_anim_set_repeat_count(&a, LV_ANIM_REPEAT_INFINITE);
    lv_anim_set_path_cb(&a, lv_anim_path_ease_in_out);

    lv_anim_set_exec_cb(&a, anim_size_cb);
    lv_anim_start(&a);
    lv_anim_set_exec_cb(&a, anim_x_cb);
    lv_anim_set_values(&a, 10, 240);
    lv_anim_start(&a);
}

#endif
```

Animazione di cubica di Bézier

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_SLIDER && LV_USE_CHART && LV_USE_BUTTON &&␣
↪LV_USE_GRID

/**
 * the example show the use of cubic-bezier3 in animation.
 * the control point P1,P2 of cubic-bezier3 can be adjusted by slider.
```

(continues on next page)

(continua dalla pagina precedente)

```
 * and the chart shows the cubic-bezier3 in real time. you can click
 * run button see animation in current point of cubic-bezier3.
 */

#define CHART_POINTS_NUM 256

struct {
    lv_obj_t * anim_obj;
    lv_obj_t * chart;
    lv_chart_series_t * ser1;
    lv_obj_t * p1_slider;
    lv_obj_t * p1_label;
    lv_obj_t * p2_slider;
    lv_obj_t * p2_label;
    lv_obj_t * run_btn;
    int32_t p1;
    int32_t p2;
    lv_anim_t a;
} ginfo;

static int32_t anim_path_bezier3_cb(const lv_anim_t * a);
static void refer_chart_cubic_bezier(void);
static void run_button_event_handler(lv_event_t * e);
static void slider_event_cb(lv_event_t * e);
static void page_obj_init(lv_obj_t * par);
static void anim_x_cb(void * var, int32_t v);

/**
 * create an animation
 */
void lv_example_anim_3(void)
{
    static int32_t col_dsc[] = {LV_GRID_FR(1), 200, LV_GRID_FR(1), LV_GRID_
↪TEMPLATE_LAST};
    static int32_t row_dsc[] = {30, 10, 10, LV_GRID_FR(1), LV_GRID_
↪TEMPLATE_LAST};

    /*Create a container with grid*/
    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_style_pad_all(cont, 2, LV_PART_MAIN);
    lv_obj_set_style_pad_column(cont, 10, LV_PART_MAIN);
    lv_obj_set_style_pad_row(cont, 10, LV_PART_MAIN);
    lv_obj_set_grid_dsc_array(cont, col_dsc, row_dsc);
    lv_obj_set_size(cont, 320, 240);
    lv_obj_center(cont);

    page_obj_init(cont);

    lv_anim_init(&ginfo.a);
    lv_anim_set_var(&ginfo.a, ginfo.anim_obj);
    int32_t end = lv_obj_get_style_width(cont, LV_PART_MAIN) -
                  lv_obj_get_style_width(ginfo.anim_obj, LV_PART_MAIN) -␣
↪10;
    lv_anim_set_values(&ginfo.a, 5, end);
    lv_anim_set_duration(&ginfo.a, 2000);
    lv_anim_set_exec_cb(&ginfo.a, anim_x_cb);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_anim_set_path_cb(&ginfo.a, anim_path_bezier3_cb);

    refer_chart_cubic_bezier();
}

static int32_t anim_path_bezier3_cb(const lv_anim_t * a)
{
    int32_t t = lv_map(a->act_time, 0, a->duration, 0, 1024);
    int32_t step = lv_bezier3(t, 0, ginfo.p1, ginfo.p2, 1024);
    int32_t new_value;
    new_value = step * (a->end_value - a->start_value);
    new_value = new_value >> 10;
    new_value += a->start_value;
    return new_value;
}

static void refer_chart_cubic_bezier(void)
{
    for(uint16_t i = 0; i <= CHART_POINTS_NUM; i ++) {
        int32_t t = i * (1024 / CHART_POINTS_NUM);
        int32_t step = lv_bezier3(t, 0, ginfo.p1, ginfo.p2, 1024);
        lv_chart_set_series_value_by_id2(ginfo.chart, ginfo.ser1, i, t,␣
↪step);
    }
    lv_chart_refresh(ginfo.chart);
}

static void anim_x_cb(void * var, int32_t v)
{
    lv_obj_set_style_translate_x((lv_obj_t *)var, v, LV_PART_MAIN);
}

static void run_button_event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    if(code == LV_EVENT_CLICKED) {
        lv_anim_start(&ginfo.a);
    }
}

static void slider_event_cb(lv_event_t * e)
{
    char buf[16];
    lv_obj_t * label;
    lv_obj_t * slider = lv_event_get_target_obj(e);

    if(slider == ginfo.p1_slider) {
        label = ginfo.p1_label;
        ginfo.p1 = lv_slider_get_value(slider);
        lv_snprintf(buf, sizeof(buf), "p1:%d", ginfo.p1);
    }
    else {
        label = ginfo.p2_label;
        ginfo.p2 = lv_slider_get_value(slider);
        lv_snprintf(buf, sizeof(buf), "p2:%d", ginfo.p2);
    }
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_label_set_text(label, buf);
    refer_chart_cubic_bezier();
}

static void page_obj_init(lv_obj_t * par)
{
    ginfo.anim_obj = lv_obj_create(par);
    lv_obj_set_size(ginfo.anim_obj, 30, 30);
    lv_obj_set_align(ginfo.anim_obj, LV_ALIGN_TOP_LEFT);
    lv_obj_remove_flag(ginfo.anim_obj, LV_OBJ_FLAG_SCROLLABLE);
    lv_obj_set_style_bg_color(ginfo.anim_obj, lv_palette_main(LV_PALETTE_
↪RED), LV_PART_MAIN);
    lv_obj_set_grid_cell(ginfo.anim_obj, LV_GRID_ALIGN_START, 0, 1, LV_
↪GRID_ALIGN_START, 0, 1);

    ginfo.p1_label = lv_label_create(par);
    ginfo.p2_label = lv_label_create(par);
    lv_label_set_text(ginfo.p1_label, "p1:0");
    lv_label_set_text(ginfo.p2_label, "p2:0");
    lv_obj_set_grid_cell(ginfo.p1_label, LV_GRID_ALIGN_START, 0, 1, LV_
↪GRID_ALIGN_START, 1, 1);
    lv_obj_set_grid_cell(ginfo.p2_label, LV_GRID_ALIGN_START, 0, 1, LV_
↪GRID_ALIGN_START, 2, 1);

    ginfo.p1_slider = lv_slider_create(par);
    ginfo.p2_slider = lv_slider_create(par);
    lv_slider_set_range(ginfo.p1_slider, 0, 1024);
    lv_slider_set_range(ginfo.p2_slider, 0, 1024);
    lv_obj_set_style_pad_all(ginfo.p1_slider, 2, LV_PART_KNOB);
    lv_obj_set_style_pad_all(ginfo.p2_slider, 2, LV_PART_KNOB);
    lv_obj_add_event_cb(ginfo.p1_slider, slider_event_cb, LV_EVENT_VALUE_
↪CHANGED, NULL);
    lv_obj_add_event_cb(ginfo.p2_slider, slider_event_cb, LV_EVENT_VALUE_
↪CHANGED, NULL);
    lv_obj_set_grid_cell(ginfo.p1_slider, LV_GRID_ALIGN_STRETCH, 1, 1, LV_
↪GRID_ALIGN_START, 1, 1);
    lv_obj_set_grid_cell(ginfo.p2_slider, LV_GRID_ALIGN_STRETCH, 1, 1, LV_
↪GRID_ALIGN_START, 2, 1);

    ginfo.run_btn = lv_button_create(par);
    lv_obj_add_event_cb(ginfo.run_btn, run_button_event_handler, LV_EVENT_
↪CLICKED, NULL);
    lv_obj_t * btn_label = lv_label_create(ginfo.run_btn);
    lv_label_set_text(btn_label, LV_SYMBOL_PLAY);
    lv_obj_center(btn_label);
    lv_obj_set_grid_cell(ginfo.run_btn, LV_GRID_ALIGN_STRETCH, 2, 1, LV_
↪GRID_ALIGN_STRETCH, 1, 2);

    ginfo.chart = lv_chart_create(par);
    lv_obj_set_style_pad_all(ginfo.chart, 0, LV_PART_MAIN);
    lv_obj_set_style_size(ginfo.chart, 0, 0, LV_PART_INDICATOR);
    lv_chart_set_type(ginfo.chart, LV_CHART_TYPE_SCATTER);
    ginfo.ser1 = lv_chart_add_series(ginfo.chart, lv_palette_main(LV_
↪PALETTE_RED), LV_CHART_AXIS_PRIMARY_Y);
    lv_chart_set_axis_range(ginfo.chart, LV_CHART_AXIS_PRIMARY_Y, 0, 1024);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_chart_set_axis_range(ginfo.chart, LV_CHART_AXIS_PRIMARY_X, 0, 1024);
    lv_chart_set_point_count(ginfo.chart, CHART_POINTS_NUM);
    lv_obj_set_grid_cell(ginfo.chart, LV_GRID_ALIGN_STRETCH, 0, 3, LV_GRID_
↪ALIGN_STRETCH, 3, 1);
}

#endif
```

Pausa animazione

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_SWITCH

static void anim_x_cb(void * var, int32_t v)
{
    lv_obj_set_x((lv_obj_t *) var, v);
}

static void timer_cb(lv_timer_t * timer)
{

    lv_anim_t * anim = (lv_anim_t *) lv_timer_get_user_data(timer);
    lv_anim_pause_for(anim, 1000);
    lv_timer_delete(timer);
}
static void sw_event_cb(lv_event_t * e)
{
    lv_obj_t * sw = lv_event_get_target_obj(e);
    lv_obj_t * label = (lv_obj_t *) lv_event_get_user_data(e);

    if(lv_obj_has_state(sw, LV_STATE_CHECKED)) {
        lv_anim_t a;
        lv_anim_init(&a);
        lv_anim_set_var(&a, label);
        lv_anim_set_values(&a, lv_obj_get_x(label), 100);
        lv_anim_set_duration(&a, 500);
        lv_anim_set_exec_cb(&a, anim_x_cb);
        lv_anim_set_path_cb(&a, lv_anim_path_overshoot);
        lv_timer_create(timer_cb, 200, lv_anim_start(&a));
    }
    else {
        lv_anim_t a;
        lv_anim_init(&a);
        lv_anim_set_var(&a, label);
        lv_anim_set_values(&a, lv_obj_get_x(label), -lv_obj_get_
↪width(label));
        lv_anim_set_duration(&a, 500);
        lv_anim_set_exec_cb(&a, anim_x_cb);
        lv_anim_set_path_cb(&a, lv_anim_path_ease_in);
        lv_timer_create(timer_cb, 200, lv_anim_start(&a));

    }

}

/**
```

(continues on next page)

(continua dalla pagina precedente)

```
 * Start animation on an event
 */
void lv_example_anim_4(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Hello animations!");
    lv_obj_set_pos(label, 100, 10);

    lv_obj_t * sw = lv_switch_create(lv_screen_active());
    lv_obj_center(sw);
    lv_obj_add_state(sw, LV_STATE_CHECKED);
    lv_obj_add_event_cb(sw, sw_event_cb, LV_EVENT_VALUE_CHANGED, label);
}

#endif
```

Timeline animazione

```
#include "../lv_examples.h"
#if LV_USE_FLEX && LV_BUILD_EXAMPLES

static const int32_t obj_width = 90;
static const int32_t obj_height = 70;

static void set_width(lv_anim_t * var, int32_t v)
{
    lv_obj_set_width((lv_obj_t *) var->var, v);
}

static void set_height(lv_anim_t * var, int32_t v)
{
    lv_obj_set_height((lv_obj_t *) var->var, v);
}

static void set_slider_value(lv_anim_t * var, int32_t v)
{
    lv_slider_set_value((lv_obj_t *) var->var, v, LV_ANIM_OFF);
}

static void btn_start_event_handler(lv_event_t * e)
{
    lv_obj_t * btn = lv_event_get_current_target_obj(e);
    lv_anim_timeline_t * anim_timeline = (lv_anim_timeline_t *) lv_event_
↪get_user_data(e);

    bool reverse = lv_obj_has_state(btn, LV_STATE_CHECKED);
    lv_anim_timeline_set_reverse(anim_timeline, reverse);
    lv_anim_timeline_start(anim_timeline);
}

static void btn_pause_event_handler(lv_event_t * e)
{
    lv_anim_timeline_t * anim_timeline = (lv_anim_timeline_t *) lv_event_
↪get_user_data(e);
    lv_anim_timeline_pause(anim_timeline);
}
```

(continues on next page)

(continua dalla pagina precedente)

```
static void slider_prg_event_handler(lv_event_t * e)
{
    lv_obj_t * slider = lv_event_get_current_target_obj(e);
    lv_anim_timeline_t * anim_timeline = (lv_anim_timeline_t *) lv_event_
↪get_user_data(e);
    int32_t progress = lv_slider_get_value(slider);
    lv_anim_timeline_set_progress(anim_timeline, (uint16_t) progress);
}

/**
 * Create an animation timeline
 */
void lv_example_anim_timeline_1(void)
{
    /* Create anim timeline */
    lv_anim_timeline_t * anim_timeline = lv_anim_timeline_create();

    lv_obj_t * par = lv_screen_active();
    lv_obj_set_flex_flow(par, LV_FLEX_FLOW_ROW);
    lv_obj_set_flex_align(par, LV_FLEX_ALIGN_SPACE_AROUND, LV_FLEX_ALIGN_
↪CENTER, LV_FLEX_ALIGN_CENTER);

    /* create btn_start */
    lv_obj_t * btn_start = lv_button_create(par);
    lv_obj_add_event_cb(btn_start, btn_start_event_handler, LV_EVENT_VALUE_
↪CHANGED, anim_timeline);
    lv_obj_add_flag(btn_start, LV_OBJ_FLAG_IGNORE_LAYOUT);
    lv_obj_add_flag(btn_start, LV_OBJ_FLAG_CHECKABLE);
    lv_obj_align(btn_start, LV_ALIGN_TOP_MID, -100, 20);

    lv_obj_t * label_start = lv_label_create(btn_start);
    lv_label_set_text(label_start, "Start");
    lv_obj_center(label_start);

    /* create btn_pause */
    lv_obj_t * btn_pause = lv_button_create(par);
    lv_obj_add_event_cb(btn_pause, btn_pause_event_handler, LV_EVENT_
↪CLICKED, anim_timeline);
    lv_obj_add_flag(btn_pause, LV_OBJ_FLAG_IGNORE_LAYOUT);
    lv_obj_align(btn_pause, LV_ALIGN_TOP_MID, 100, 20);

    lv_obj_t * label_pause = lv_label_create(btn_pause);
    lv_label_set_text(label_pause, "Pause");
    lv_obj_center(label_pause);

    /* create slider_prg */
    lv_obj_t * slider_prg = lv_slider_create(par);
    lv_obj_add_event_cb(slider_prg, slider_prg_event_handler, LV_EVENT_
↪VALUE_CHANGED, anim_timeline);
    lv_obj_add_flag(slider_prg, LV_OBJ_FLAG_IGNORE_LAYOUT);
    lv_obj_align(slider_prg, LV_ALIGN_BOTTOM_MID, 0, -20);
    lv_slider_set_range(slider_prg, 0, LV_ANIM_TIMELINE_PROGRESS_MAX);

    /* create 3 objects */
    lv_obj_t * obj1 = lv_obj_create(par);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_size(obj1, obj_width, obj_height);
    lv_obj_set_scrollbar_mode(obj1, LV_SCROLLBAR_MODE_OFF);

    lv_obj_t * obj2 = lv_obj_create(par);
    lv_obj_set_size(obj2, obj_width, obj_height);
    lv_obj_set_scrollbar_mode(obj2, LV_SCROLLBAR_MODE_OFF);

    lv_obj_t * obj3 = lv_obj_create(par);
    lv_obj_set_size(obj3, obj_width, obj_height);
    lv_obj_set_scrollbar_mode(obj3, LV_SCROLLBAR_MODE_OFF);

    /* anim-slider */
    lv_anim_t a_slider;
    lv_anim_init(&a_slider);
    lv_anim_set_var(&a_slider, slider_prg);
    lv_anim_set_values(&a_slider, 0, LV_ANIM_TIMELINE_PROGRESS_MAX);
    lv_anim_set_custom_exec_cb(&a_slider, set_slider_value);
    lv_anim_set_path_cb(&a_slider, lv_anim_path_linear);
    lv_anim_set_duration(&a_slider, 700);

    /* anim-obj1 */
    lv_anim_t a1;
    lv_anim_init(&a1);
    lv_anim_set_var(&a1, obj1);
    lv_anim_set_values(&a1, 0, obj_width);
    lv_anim_set_custom_exec_cb(&a1, set_width);
    lv_anim_set_path_cb(&a1, lv_anim_path_overshoot);
    lv_anim_set_duration(&a1, 300);

    lv_anim_t a2;
    lv_anim_init(&a2);
    lv_anim_set_var(&a2, obj1);
    lv_anim_set_values(&a2, 0, obj_height);
    lv_anim_set_custom_exec_cb(&a2, set_height);
    lv_anim_set_path_cb(&a2, lv_anim_path_ease_out);
    lv_anim_set_duration(&a2, 300);

    /* anim-obj2 */
    lv_anim_t a3;
    lv_anim_init(&a3);
    lv_anim_set_var(&a3, obj2);
    lv_anim_set_values(&a3, 0, obj_width);
    lv_anim_set_custom_exec_cb(&a3, set_width);
    lv_anim_set_path_cb(&a3, lv_anim_path_overshoot);
    lv_anim_set_duration(&a3, 300);

    lv_anim_t a4;
    lv_anim_init(&a4);
    lv_anim_set_var(&a4, obj2);
    lv_anim_set_values(&a4, 0, obj_height);
    lv_anim_set_custom_exec_cb(&a4, set_height);
    lv_anim_set_path_cb(&a4, lv_anim_path_ease_out);
    lv_anim_set_duration(&a4, 300);

    /* anim-obj3 */
    lv_anim_t a5;
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_anim_init(&a5);
    lv_anim_set_var(&a5, obj3);
    lv_anim_set_values(&a5, 0, obj_width);
    lv_anim_set_custom_exec_cb(&a5, set_width);
    lv_anim_set_path_cb(&a5, lv_anim_path_overshoot);
    lv_anim_set_duration(&a5, 300);

    lv_anim_t a6;
    lv_anim_init(&a6);
    lv_anim_set_var(&a6, obj3);
    lv_anim_set_values(&a6, 0, obj_height);
    lv_anim_set_custom_exec_cb(&a6, set_height);
    lv_anim_set_path_cb(&a6, lv_anim_path_ease_out);
    lv_anim_set_duration(&a6, 300);

    /* add animations to timeline */
    lv_anim_timeline_add(anim_timeline, 0, &a_slider);
    lv_anim_timeline_add(anim_timeline, 0, &a1);
    lv_anim_timeline_add(anim_timeline, 0, &a2);
    lv_anim_timeline_add(anim_timeline, 200, &a3);
    lv_anim_timeline_add(anim_timeline, 200, &a4);
    lv_anim_timeline_add(anim_timeline, 400, &a5);
    lv_anim_timeline_add(anim_timeline, 400, &a6);

    lv_anim_timeline_set_progress(anim_timeline, LV_ANIM_TIMELINE_PROGRESS_
↪MAX);
}

#endif
```

7.7.10 API

7.8 File System (lv_fs_drv)

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

> **Nota**
>
> If you want to skip the drive-letter prefix in Unix-like paths, you can use the *`LV_FS_DEFAULT_DRIVER_LETTER`* config parameter.

7.8.1 Ready-to-Use Drivers

LVGL contains prepared drivers for the API of POSIX, standard C, Windows, and FATFS. Learn more *here*.

7.8.2 Identifier Letters

As mentioned above, a file system is identified by an assigned identifier letter. This identifier is merely a way for the LVGL File System abstraction logic to look up the appropriate registered file-system driver for a given path.

How it Works:

You register a driver for your file system and assign it an identifier letter. This letter must be unique among all registered file-system drivers, and in the range [A-Z] or the character '/'. See *Adding a Driver* for how this is done.

Later, when using paths to files on your file system, you prefix the path with that identifier character plus a colon (':').

> **Nota**
>
> Do not confuse this with a Windows or DOS drive letter.

Example:

Let's use the letter 'Z' as the identifier character, and "path_to_file" as the path, then the path strings you pass to `lv_fs_...()` functions would look like this:

```
"Z:path_to_file"
 ^ ^^^^^^^^^^^^
 |        |
 |        +-- This part gets passed to the OS-level file-system functions.
 |
 +-- This part LVGL strips from path string, and uses it to find the␣
↪appropriate
    driver (i.e. set of functions) that apply to that file system.
```

Note also that the path can be a relative path or a "rooted path" (beginning with `/`), though rooted paths are recommended since the driver does not yet provide a way to set the default directory.

Examples for Unix-like file systems:

- "Z:/etc/images/splash.png"
- "Z:/etc/images/left_button.png"
- "Z:/etc/images/right_button.png"
- "Z:/home/users/me/wip/proposal.txt"

Examples for Windows/DOS-like file systems:

- "Z:C:/Users/me/wip/proposal.txt"
- "Z:/Users/me/wip/proposal.txt" (if the default drive is known to be C:)
- "Z:C:/Users/Public/Documents/meeting_notes.txt"
- "Z:D:/to_print.docx"

Reminder: Note carefully that the prefixed "Z:" has nothing to do with the "C:" and "D:" Windows/DOS drive letters in 3 of the above examples, which are part of the path. "Z:" is used to look up the driver for that file system in the list of all file-system drivers registered with LVGL.

7.8.3 Adding a Driver

Registering a driver

To add a driver, a *`lv_fs_drv_t`* object needs to be initialized and registered in a way similar to the code below. The *`lv_fs_drv_t`* variable needs to be static, global or dynamically allocated and not a local variable, since its contents need to remain valid as long as the driver is in use.

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

drv.letter = 'S';                           /* An uppercase letter to␣
↪identify the drive */
drv.cache_size = my_cache_size;             /* Cache size for reading in␣
↪bytes. 0 to not cache. */
```

(continues on next page)

(continua dalla pagina precedente)

```
drv.ready_cb = my_ready_cb;                /* Callback to tell if the drive␣
↪is ready to use */
drv.open_cb = my_open_cb;                  /* Callback to open a file */
drv.close_cb = my_close_cb;                /* Callback to close a file */
drv.read_cb = my_read_cb;                  /* Callback to read a file */
drv.write_cb = my_write_cb;                /* Callback to write a file */
drv.seek_cb = my_seek_cb;                  /* Callback to seek in a file␣
↪(Move cursor) */
drv.tell_cb = my_tell_cb;                  /* Callback to tell the cursor␣
↪position  */

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

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

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

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

Implementing the callbacks

Open callback

The prototype of `open_cb` looks like this:

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

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

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

Other callbacks

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

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

For `file_p`, LVGL passes the return value of `open_cb`, `buf` is the data to write, `btw` is the number of "bytes to write", `bw` is the number of "bytes written" (written to during the function call).

For a list of prototypes for these callbacks see lv_fs_template.c. This file also provides a template for new file-system drivers you can use if the one you need is not already provided.

Drivers that come with LVGL

As of this writing, the list of already-available file-system drivers can be enabled by setting one or more of the following macros to a non-zero value in `lv_conf.h`. The drivers are as implied by the macro names.

If you use more than one, each associated identifier letter you use must be unique.

- *LV_USE_FS_FATFS*

- *LV_USE_FS_STDIO*
- *LV_USE_FS_POSIX*
- *LV_USE_FS_WIN32*
- *LV_USE_FS_MEMFS*
- *LV_USE_FS_LITTLEFS*
- *LV_USE_FS_ARDUINO_ESP_LITTLEFS*
- *LV_USE_FS_ARDUINO_SD*

7.8.4 Limiting Directory Access

If you are using one of the following file-system drivers:

- *LV_USE_FS_STDIO*
- *LV_USE_FS_POSIX*
- *LV_USE_FS_WIN32*

you will have a `LV_FS_xxx_PATH` macro available to you in `lv_conf.h` that you can use to provide a path that gets dynamically prefixed to the `path_to_file` portion of of the path strings provided to `lv_fs_...()` functions when files and directories are opened. This can be useful to limit directory access (e.g. when a portion of a path can be typed by an end user), or simply to reduce the length of the path strings provided to `lv_fs_...()` functions.

Do this by filling in the full path to the directory you wish his access to be limited to in the applicable `LV_FS_xxx_PATH` macro in `lv_conf.h`. Do not prefix the path with the driver-identifier letter, and do append a directory separator character at the end.

Examples for Unix-like file systems:

```
#define LV_FS_WIN32_PATH   "/home/users/me/"
```

Examples for Windows/DOS-like file systems:

```
#define LV_FS_WIN32_PATH   "C:/Users/me/"
```

Then in both cases, path strings passed to `lv_fs_...()` functions in the application get reduced to:

- "Z:wip/proposal.txt"

7.8.5 Usage Example

The example below shows how to read from a file:

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

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

lv_fs_close(&f);
```

The mode in *lv_fs_open()* can be *LV_FS_MODE_WR* to open for writes only, *LV_FS_MODE_RD* for reads only, or *LV_FS_MODE_RD* | *LV_FS_MODE_WR* for both.

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

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

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

    /* fn is empty if there are no more files to read. */
    if(strlen(fn) == 0) {
        break;
    }

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

lv_fs_dir_close(&dir);
```

7.8.6 Use Drives for Images

Image Widgets can be opened from files as well (besides variables stored in the compiled program).

To use files in Image Widgets the following callbacks are required:

- open
- close
- read
- seek
- tell

7.8.7 Optional File Buffering/Caching

Files will buffer their reads if the corresponding `LV_FS_*_CACHE_SIZE` config option is set to a value greater than zero. Each open file will buffer up to that many bytes to reduce the number of FS driver calls.

Generally speaking, file buffering can be optimized for different kinds of access patterns. The one implemented here is optimal for reading large files in chunks, which is what the image decoder does. It has the potential to call the driver's `read` fewer times than `lv_fs_read` is called. In the best case where the cache size is >= the size of the file, `read` will only be called once. This strategy is good for linear reading of large files but less helpful for short random reads across a file bigger than the buffer since data will be buffered that will be discarded after the next seek and read. The cache should be sufficiently large or disabled in that case. Another case where the cache should be disabled is if the file contents are expected to change by an external factor like with special OS files.

The implementation is documented below. Note that the FS functions make calls to other driver FS functions when the cache is enabled. i.e., `lv_fs_read` may call the driver's `seek` so the driver needs to implement more callbacks when the cache is enabled.

`lv_fs_read` (behavior when cache is enabled)

(Use mouse wheel to zoom in, and drag image to pan it.)

`lv_fs_write` (behavior when cache is enabled)

The part of the cache that coincides with the written content will be updated to reflect the written content.

`lv_fs_seek` (behavior when cache is enabled)

The driver's `seek` will not actually be called unless the `whence` is `LV_FS_SEEK_END`, in which case `seek` and `tell` will be called to determine where the end of the file is.

`lv_fs_tell` (behavior when cache is enabled)

The driver's `tell` will not actually be called.

7.8.8 API

lv_fs.h

lv_fs_private.h

lv_fsdrv.h

7.9 Drawing

7.9.1 Draw Pipeline

What is Drawing?

Drawing (also known as *rendering*) is writing pixel colors into a buffer where they will be delivered to a display panel as pixels. Sometimes this is done by copying colors from places like background- and foreground-color properties. Other times it involves computing those colors before they are written (e.g. combining them with other colors when an object higher on the Z axis has partial opacity).

The following sections cover the LVGL drawing logic and how to use it and optionally tune it to fit your particular project (e.g. if you have a GPU or other resources that you would like to get involved).

Draw-Pipeline Overview

On modern computing hardware meant to be used with larger display panels, there are sometimes options for different ways drawing can be accomplished. For example, some MCUs come with hardware that is very good (and fast) at certain types of drawing tasks. Alternatively, you might have access to a drawing library that performs certain types of drawing tasks with great efficiency. To make it possible to utilize such facilities in the most efficient fashion, LVGL v9 and onwards implements a *Drawing Pipeline*, like an assembly line, where decisions are made as to which drawing tasks (*Draw Tasks*) are given to which "logic entities" (*Draw Units*) in order to be carried out.

This Pipeline is designed so that it is both flexible and extendable. You can use it to perform custom rendering with a GPU, or replace the parts of the built-in software rendering logic to any extent desired.

Using events, it's also possible to modify *Draw Tasks* or insert new ones as LVGL renders Widgets.

The following sections describe the basic terminology and concepts of rendering.

Draw Tasks

A "Draw Task" (`lv_draw_task_t`) is a package of information that is created at the beginning of the Drawing Pipeline when a request to draw is made. Functions such as `lv_draw_rect()` and `lv_draw_label()` create one or more Draw Tasks and pass them down the Drawing Pipeline. Each Draw Task carries all the information required to:

- compute which *Draw Unit* should receive this task, and
- give the Draw Unit all the information required to accomplish the drawing task.

A Draw Task carries the following information:

type
defines the drawing algorithm involved (e.g. line, fill, border, image, label, arc, triangle, etc.)

area
defines the rectangle in which drawing will occur

transformation matrix
if `LV_DRAW_TRANSFORM_USE_MATRIX` is configured to '1'

state
waiting, queued, in progress, completed

drawing descriptor
carries details of the drawing to be performed

preferred Draw Unit ID
identifier of the Draw Unit that should carry out this task

preference score
value describing the speed of the specified Draw Unit relative to software rendering (more on this below)

next
a link to the next Draw Task in the list.

Draw Tasks are collected in a list and periodically dispatched to Draw Units.

Draw Units

A "Draw Unit" (based on `lv_draw_unit_t`) is any "logic entity" that can generate the output required by a *Draw Task*. This can be a CPU core, a GPU, a custom rendering library for specific Draw Tasks, or any entity capable of performing rendering.

For a reference implementation of a draw unit, see lv_draw_sw.c.

During LVGL's initialization (`lv_init()`), a list of Draw Units is created. If `LV_USE_DRAW_SW` is set to `1` in `lv_conf.h` (it is by default), the Software Drawing Unit enters itself at the head of that list. If your platform has other drawing units available, if they are configured to be used in `lv_conf.h`, they are added to this list during LVGL's initialization. If you are adding your own Draw Unit(s), you add each available drawing unit to that list by calling `lv_draw_create_unit(sizeof(your_draw_unit_t))`. With each call to that function, the newly-created draw unit is added to the head of that list, pushing already-existing draw units further back in the list, pushing the Draw Units created earlier farther back in the list. The order of this list (and thus the order in which *Draw Task Evaluation* is performed) is governed by the order in which each Draw Unit is created.

Building this list (and initializing the Draw Units) is normally handled automatically by configuring the available Draw Units in `lv_conf.h`, such as setting `LV_USE_DRAW_OPENGLES` or `LV_USE_PXP` or `LV_USE_DRAW_SDL` or `LV_USE_DRAW_VG_LITE` to `1`. However, if you are introducing your own Draw Unit(s), you will need to create and initialize it (after `lv_init()`) as above. This will include several things, but setting its `evaluate_cb` and `dispatch_cb` callbacks (mentioned later) are two of them.

For an example of how draw-unit creation and initialization is done, see `lv_draw_sw_init()` in lv_draw_sw.c or the other draw units whose `init` functions are optionally called in `lv_init()`.

Thread Priority

All draw units operate with a configurable thread priority which can be set using the `LV_DRAW_THREAD_PRIO` configuration option in `lv_conf.h`. This allows you to fine-tune the priority level across all drawing units, which is especially useful for systems with limited priority levels.

By default, draw units use `LV_THREAD_PRIO_HIGH` as their thread priority. This consistent approach ensures that all drawing units (software rendering, hardware accelerators like STM32 DMA2D, NXP VGLite, etc.) use the same priority level unless explicitly configured otherwise.

Draw Task Evaluation

When each *Draw Task* is created, each existing Draw Unit is "consulted" as to its "appropriateness" for the task. It does this through an "evaluation callback" function pointer (a.k.a. `evaluate_cb`), which each Draw Unit sets (for itself) during its initialization. Normally, that evaluation:

- optionally examines the existing "preference score" for the task mentioned above,
- if it can accomplish that type of task (e.g. line drawing) faster than other Draw Units that have already reported, it writes its own "preference score" and "preferred Draw Unit ID" to the respective fields in the task.

In this way, by the time the evaluation sequence is complete, the task will contain the score and the ID of the Drawing Unit that will be used to perform that task when it is *dispatched*.

This logic, of course, can be overridden or redefined, depending on system design.

As a side effect, this also ensures that the same Draw Unit will be selected consistently, depending on the type (and nature) of the drawing task, avoiding any possible screen jitter in case more than one Draw Unit is capable of performing a given task type.

The sequence of the Draw Unit list (with the Software Draw Unit at the end) also ensures that the Software Draw Unit is the "buck-stops-here" Draw Unit: if no other Draw Unit reported it was better at a given drawing task, then the Software Draw Unit will handle it.

Dispatching

While collecting Draw Tasks LVGL frequently dispatches the collected Draw Tasks to their assigned Draw Units. This is handled via the `dispatch_cb` of the Draw Units.

If a Draw Unit is busy with another Draw Task, it just returns. However, if it is available it can take a Draw Task.

`lv_draw_get_next_available_task``(layer, previous_task, draw_unit_id)` is a useful helper function which is used by the `dispatch_cb` to get the next Draw Task it should act on. If it handled the task, it sets the Draw Task's `state` field to *`LV_DRAW_TASK_STATE_FINISHED`*. "Available" in this context means that has been queued and assigned to a given Draw Unit and is ready to be carried out. The ramifications of having multiple drawing threads are taken into account for this.

Run-Time Object Hierarchy

All of the above have this relationship at run time:

- LVGL (global)
  - list of *Draw Units*
  - list of *Display(s)*
    - Layer(s): Each *Display object* has its own list of *Draw Layers*
      - Draw Tasks: Each Layer has its own list of *Draw Tasks*

API

lv_conf_internal.h

lv_draw.h

lv_draw_label.h

lv_draw_private.h

lv_draw_rect.h

lv_draw_sw.h

7.9.2 Draw API

Where to Use the Drawing API

In most cases you use LVGL's Drawing API through the API of Widgets: by creating buttons, labels, etc., and setting the their styles, positions, and other properties. In these cases rendering (drawing) is handled internally and you don't see the *Drawing Pipeline* at all.

However there are three places where you can use LVGL's Drawing API directly.

1. **In the draw events of the Widgets**: There are event codes which are sent when the Widget needs to render itself:

   - `LV_EVENT_DRAW_MAIN_BEGIN`, `LV_EVENT_DRAW_MAIN`, `LV_EVENT_DRAW_MAIN_END`: Triggered before, during, and after a Widget is drawn, respectively. Widget rendering typically occurs in `LV_EVENT_DRAW_MAIN`.
   - `LV_EVENT_DRAW_POST_BEGIN`, `LV_EVENT_DRAW_POST`, `LV_EVENT_DRAW_POST_END`: Triggered before, during, and after all child Widgets are rendered, respectively. This can be useful for overlay-like drawings, such as scrollbars which should be rendered on top of any children.

   These are relevant if a new Widget is implemented and it uses custom drawing.

2. **Modifying the created draw tasks**: The when a draw task is created for a Widget `LV_EVENT_DRAW_TASK_ADDED` is sent. In this event the created draw task can be modified or new draw tasks can be added. Typical use cases for this are modifying each bar of a bar chart, or cells of a table.

   For performance reasons, this event is disabled by default. Enable it by setting the `LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS` flag on the Widget(s) you wish to emit this event.

3. **Draw to the Canvas Widget**: The drawing functions can be used directly to draw to a Canvas Widget. Doing so renders custom drawing to a buffer which can be used later as an image or a mask.

   For more information see *Canvas (lv_canvas)*.

Drawing API

The main components of LVGL's Drawing API are the `lv_draw_rect()`, `lv_draw_label()`, `lv_draw_image()`, and similar functions. When they are called `lv_draw_task_t` objects are created internally.

These functions have the following parameters:

- **Layer**: This is the target of the drawing. See details at *Draw Layers*.
- **Draw Descriptor**: This is a large `struct` containing all the information about the drawing. See details of the draw descriptors at *Draw Descriptors*.
- **Area** (in some cases): Specifies where to draw.

Coordinate System

Some functions and draw descriptors require area or point parameters. These are always **absolute coordinates** on the display. For example, if the target layer is on a 800x480 display and the layer's area is (100,100) (200,200), to render a 10x10 object in the middle, the coordinates (145,145) (154,154) should be used (not (40,40) (49,49)).

Exception: for the Canvas Widget the layer is always assumed to be at the (0,0) coordinate, regardless of the Canvas Widget's position.

API

lv_draw_arc.h

lv_draw_image.h

lv_draw_label.h

lv_draw_line.h

lv_draw_mask.h

lv_draw_triangle.h

7.9.3 Draw Layers

Not to be confused with a *Display's main 4 layers*, a *Draw Layer* is a buffer created during rendering, necessitated by certain style properties, so different sets of pixels are correctly combined. Factors requiring such layers are:

- partial opacity
- bit-mask being applied
- blend mode
- clipped corners (a bit-mask application)
- transformations
  - scale
  - skew
  - rotation

Later that layer will be merged to the screen or its parent layer at the correct point in the rendering sequence.

Layer Types

Simple Layer

The following style properties trigger the creation of a "Simple Layer":

- `opa_layered`
- `bitmap_mask_src`
- `blend_mode`

In this case, the Widget will be sliced into `LV_DRAW_SW_LAYER_SIMPLE_BUF_SIZE` sized chunks.

If there is no memory for a new chunk, LVGL will try allocating the layer after another chunk is rendered and freed.

Transform Layer

The following style properties trigger the creation of a "Transform Layer":

- `transform_scale_x`
- `transform_scale_y`
- `transform_skew_x`
- `transform_skew_y`
- `transform_rotate`

Due to the nature of transformations, the Widget being transformed (and its children) must be rendered first, followed by the transformation step. This necessitates a temporary drawing area (layer), often larger than the Widget proper, to provide an area of adequate size for the transformation. LVGL tries to render as small area of the widget as possible, but due to the nature of transformations no slicing is possible in this case.

Clip Corner

The `clip_corner` style property also causes LVGL to create a 2 layers with radius height for the top and bottom parts of the Widget.

Getting the Current Layer

The first parameter of the `lv_draw_rect/label/etc` functions is a layer.

In most cases a layer is not created, but an existing layer is used to draw there.

The draw API can be used in these cases and the current layer can be used differently in each case:

1. **In draw events**: In `LV_EVENT_DRAW_MAIN/POST_BEGIN/...` events the Widget is being rendered to a layer of the display or another temporary layer created earlier during rendering. The current target layer can be retrieved using *`lv_event_get_layer`*`(e)`.

   It also possible to create new layers in these events, but the previous layer is also required since it will be the parent layer in *`lv_draw_layer()`*.

2. **Modifying the created Draw Tasks**: In *`LV_EVENT_DRAW_TASK_ADDED`* the draw tasks created by `lv_draw_rect/label/etc` can be modified. It's not required to know the current layer to modify a draw task. However, if something new also needs to be drawn with `lv_draw_rect/label/etc` the current layer is also required.

   The current layer can be read from the `base` draw descriptor. Ad esempio:

   ```
   /* In LV_EVENT_DRAW_TASK_ADDED */
   lv_draw_task_t * t = lv_event_get_draw_task(e);
   lv_draw_base_dsc_t * draw_dsc = lv_draw_task_get_draw_dsc(t);

   lv_layer_t * current_layer = draw_dsc.layer;
   ```

3. **Draw to the Canvas Widget**: The canvas itself doesn't store a layer, but one can be easily created and used like this:

   ```
   /* Initialize a layer */
   lv_layer_t layer;
   lv_canvas_init_layer(canvas, &layer);

   /* Draw something here */

   /* Wait until the rendering is ready */
   lv_canvas_finish_layer(canvas, &layer);
   ```

Creating a New Layer

To create a new layer, use *`lv_draw_layer_create()`*:

```
lv_area_t layer_area = {10, 10, 80, 50}; /* Area of the new layer */
lv_layer_t * new_layer = lv_draw_layer_create(parent_layer, LV_COLOR_
↪FORMAT_RGB565, &layer_area);
```

Once the layer is created, draw tasks can be added to it by using the *Draw API* and *Draw descriptors*. In most cases this means calling the `lv_draw_rect/label/etc` functions.

Finally, the layer must be rendered to its parent layer. Since a layer behaves similarly to an image, it can be rendered the same way as images:

```
lv_draw_image_dsc_t image_draw_dsc;
lv_draw_image_dsc_init(&image_draw_dsc);
image_draw_dsc.src = new_layer; /* Source image is the new layer. */
/* Draw new layer to parent layer. */
lv_draw_layer(parent_layer, &image_draw_dsc, &layer_area);
```

Memory Considerations

Layer Buffers

The buffer for a layer (where rendering occurs) is not allocated at creation. Instead, it is allocated by *Draw Units* when the first *Draw Task* is dispatched.

Layer buffers can be large, so ensure there is sufficient heap memory or increase `LV_MEM_SIZE` in `lv_conf.h`.

Layer Type Memory Requirements

To save memory, LVGL can render certain types of layers in smaller chunks:

1. **Simple Layers**: Simple layers can be rendered in chunks. For example, with `opa_layered = 140`, only 10 lines of the layer can be rendered at a time, then the next 10 lines, and so on. This avoids allocating a large buffer for the entire layer. The buffer size for a chunk is set using `LV_DRAW_LAYER_SIMPLE_BUF_SIZE` in `lv_conf.h`.
2. **Transform Layers**: Transform Widgets cannot be rendered in chunks because transformations often affect pixels outside the given area. For such layers, LVGL allocates a buffer large enough to render the entire transformed area without limits.

Memory Limit for Layers

The total memory available for layers at once is controlled by `LV_DRAW_LAYER_MAX_MEMORY` in `lv_conf.h`. If set to `0`, there is no limit.

API

lv_draw.h

lv_obj_event.h

7.9.4 Draw Descriptors

Panoramica

Each *Draw Task* type has its own draw descriptor type. For example, `lv_draw_label_dsc_t` is used for label drawing, `lv_draw_image_dsc_t` is used for image drawing.

When an `lv_draw_...` function is called, it creates a Draw Task, copies the draw descriptor into a `malloc`ed memory block, and frees it automatically when needed. Therefore, local draw descriptor variables can be safely used.

Relation to Styles

In most cases, style properties map 1-to-1 to draw descriptor fields. Ad esempio:

- `label_dsc.color` corresponds to the `text_color` style property.
- `shadow_dsc.width`, `line_dsc.opa`, and `arc_dsc.width` map to `shadow_width`, `line_opa`, and `arc_width` in styles.

See *Style Properties* to see the list of style properties and what they mean.

Base Draw Descriptor

In each draw descriptor there is a generic "base descriptor" with `lv_draw_dsc_base_t` type and with `base` in its name. For example `label_dsc.base`. This `struct` stores useful information about which Widget and part created the draw descriptor. See all the fields in `lv_draw_dsc_base_t`.

In an `LV_EVENT_DRAW_TASK_ADDED` event, the elements of the base draw descriptor are very useful to identify the Draw Task. Ad esempio:

```
/* In LV_EVENT_DRAW_TASK_ADDED */
lv_draw_task_t * t = lv_event_get_draw_task(e);
lv_draw_base_dsc_t * draw_dsc = lv_draw_task_get_draw_dsc(t);
draw_dsc.obj;  /* The Widget for which the draw descriptor was created */
draw_dsc.part; /* The Widget part for which the draw descriptor was created
                  E.g. LV_PART_INDICATOR */
draw_dsc.id1;  /* A Widget type specific ID (e.g. table row index).
                  See the docs of the given Widget. */
draw_dsc.id2;

draw_dsc.layer; /* The target layer.
                   Required when a new Draw Tasks are also created */
```

Simple Initialization

Before using a draw descriptor it needs to be initialized with the related function. For example, `lv_draw_label_dsc_init(&my_label_draw_dsc)`.

After initialization, each field of the draw descriptor can be set. The default values are quite sane and reasonable, so usually only a few fields need modification. Ad esempio:

```
/* In LV_EVENT_DRAW_MAIN */
lv_draw_label_dsc_t my_label_draw_dsc;
lv_draw_label_dsc_init(&my_label_draw_dsc);
my_label_draw_dsc.font = &my_font;
my_label_draw_dsc.color = lv_color_hex(0xff0000);
my_label_draw_dsc.text = "Hello";

lv_area_t a = {10, 10, 200, 50}; /* Draw label here */

lv_draw_label(lv_event_get_layer(e), &my_label_draw_dsc, &a);
```

Initialization for Widgets

When rendering a part of a Widget, helper functions can initialize draw descriptors based on the styles, and a specific Widget part in the current state.

Ad esempio:

```
/* In LV_EVENT_DRAW_MAIN */
lv_draw_rect_dsc_t cur_dsc;
lv_draw_rect_dsc_init(&cur_dsc);
lv_obj_init_draw_rect_dsc(obj, LV_PART_CURSOR, &cur_dsc);
cur_dsc.fill_color = lv_color_hex(0xff0000); /* Modify if needed */
lv_draw_rect(layer, &cur_dsc, &area);
```

The `lv_obj_init_draw_...` functions automatically initialize the fields of the base descriptor.

Modify the draw descriptors

In *LV_EVENT_DRAW_TASK_ADDED*, the draw descriptor of the `draw_task` can be accessed (using *lv_draw_task_get_label_dsc()* and similar functions) and modified (to change color, text, font, etc.). This means that in *LV_EVENT_DRAW_TASK_ADDED*, the `draw_task`s and draw descriptors are already initialized and it's enough to change only a few specific values.

Ad esempio:

```
/* In LV_EVENT_DRAW_TASK_ADDED */
lv_draw_task_t * t = lv_event_get_draw_task(e);
lv_draw_label_dsc_t * draw_dsc = lv_draw_task_get_label_dsc(t);

/* Check a few things in `draw_dsc->base` */

/* Make the color lighter for longer texts */
draw_dsc->color = lv_color_lighten(draw_dsc->color,
                                   LV_MIN(lv_strlen(draw_dsc->text) * 5,␣
↪255));

/* Create new Draw Tasks if needed by calling
 * `lv_draw_...(draw_dsc->base.layer, ...)` functions */
```

Rectangle Draw Descriptor

lv_draw_rect_dsc_t is a helper descriptor that combines:

- Fill
- Border
- Outline (a border with its own styles)
- Shadow
- Background image (an image with its own styles)

into a single call.

`lv_obj_init_draw_rect_dsc(obj, part, &dsc);` initializes a draw descriptor from a Widget, and `lv_draw_rect(layer, &dsc, area)` draws the rectangle in a specified area.

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  50
#define CANVAS_HEIGHT  50

/**
 * Draw a rectangle to the canvas
 */
void lv_example_canvas_3(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf);

    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_draw_rect_dsc_t dsc;
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_draw_rect_dsc_init(&dsc);
    dsc.bg_color = lv_palette_main(LV_PALETTE_RED);
    dsc.border_color = lv_palette_main(LV_PALETTE_BLUE);
    dsc.border_width = 3;
    dsc.outline_color = lv_palette_main(LV_PALETTE_GREEN);
    dsc.outline_width = 2;
    dsc.outline_pad = 2;
    dsc.outline_opa = LV_OPA_50;
    dsc.radius = 5;
    dsc.border_width = 3;

    lv_area_t coords = {10, 10, 40, 30};

    lv_draw_rect(&layer, &dsc, &coords);

    lv_canvas_finish_layer(canvas, &layer);
}
#endif
```

Fill Draw Descriptor

The main fields of *lv_draw_fill_dsc_t* are straightforward. It has a radius, opacity, and color to draw a rectangle. If opacity is 0, no draw task will be created.

- *lv_draw_fill_dsc_init*(`&dsc`) initializes a fill Draw Task.
- *lv_draw_sw_fill*(`layer, &dsc, area`) creates a Draw Task to fill an area.
- *lv_draw_task_get_fill_dsc*(`draw_task`) retrieves the fill descriptor from a Draw Task.

Gradients

The `grad` field of the fill descriptor (or *lv_grad_dsc_t* in general) supports:

- Horizontal
- Vertical
- Skew
- Radial
- Conical

gradient types.

The following show some example gradients.

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

/**
 * Using the background style properties
 */
void lv_example_style_2(void)
{
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_radius(&style, 5);

    /*Make a gradient*/
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    static lv_grad_dsc_t grad;
    grad.dir = LV_GRAD_DIR_VER;
    grad.stops_count = 2;
    grad.stops[0].color = lv_palette_lighten(LV_PALETTE_GREY, 1);
    grad.stops[0].opa = LV_OPA_COVER;
    grad.stops[1].color = lv_palette_main(LV_PALETTE_BLUE);
    grad.stops[1].opa = LV_OPA_COVER;

    /*Shift the gradient to the bottom*/
    grad.stops[0].frac  = 128;
    grad.stops[1].frac  = 192;

    lv_style_set_bg_grad(&style, &grad);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_center(obj);
}

#endif
```

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

#if LV_USE_DRAW_SW_COMPLEX_GRADIENTS

/**
 * Simulate metallic knob using conical gradient
 * For best effect set LV_GRADIENT_MAX_STOPS to 8 or at least 3
 */
void lv_example_style_16(void)
{
#if LV_GRADIENT_MAX_STOPS >= 8
    static const lv_color_t grad_colors[8] = {
        LV_COLOR_MAKE(0xe8, 0xe8, 0xe8),
        LV_COLOR_MAKE(0xff, 0xff, 0xff),
        LV_COLOR_MAKE(0xfa, 0xfa, 0xfa),
        LV_COLOR_MAKE(0x79, 0x79, 0x79),
        LV_COLOR_MAKE(0x48, 0x48, 0x48),
        LV_COLOR_MAKE(0x4b, 0x4b, 0x4b),
        LV_COLOR_MAKE(0x70, 0x70, 0x70),
        LV_COLOR_MAKE(0xe8, 0xe8, 0xe8),
    };
#elif LV_GRADIENT_MAX_STOPS >= 3
    static const lv_color_t grad_colors[3] = {
        LV_COLOR_MAKE(0xe8, 0xe8, 0xe8),
        LV_COLOR_MAKE(0xff, 0xff, 0xff),
        LV_COLOR_MAKE(0x79, 0x79, 0x79),
    };
#else
    static const lv_color_t grad_colors[2] = {
        LV_COLOR_MAKE(0xe8, 0xe8, 0xe8),
        LV_COLOR_MAKE(0x79, 0x79, 0x79),
    };
```

(continues on next page)

(continua dalla pagina precedente)

```
#endif

    /*Create a style with gradient background and shadow*/
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_radius(&style, 500);
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_shadow_color(&style, lv_color_black());
    lv_style_set_shadow_width(&style, 50);
    lv_style_set_shadow_offset_x(&style, 20);
    lv_style_set_shadow_offset_y(&style, 20);
    lv_style_set_shadow_opa(&style, LV_OPA_50);

    /*First define a color gradient. In this example we use a gray color␣
↪map with random values.*/
    static lv_grad_dsc_t grad;

    lv_grad_init_stops(&grad, grad_colors, NULL, NULL, sizeof(grad_colors)␣
↪/ sizeof(lv_color_t));

    /*Make a conical gradient with the center in the middle of the object*/
#if LV_GRADIENT_MAX_STOPS >= 8
    lv_grad_conical_init(&grad, LV_GRAD_CENTER, LV_GRAD_CENTER, 0, 120, LV_
↪GRAD_EXTEND_REFLECT);
#elif LV_GRADIENT_MAX_STOPS >= 3
    lv_grad_conical_init(&grad, LV_GRAD_CENTER, LV_GRAD_CENTER, 45, 125,␣
↪LV_GRAD_EXTEND_REFLECT);
#else
    lv_grad_conical_init(&grad, LV_GRAD_CENTER, LV_GRAD_CENTER, 45, 110,␣
↪LV_GRAD_EXTEND_REFLECT);
#endif

    /*Set gradient as background*/
    lv_style_set_bg_grad(&style, &grad);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_set_size(obj, 200, 200);
    lv_obj_center(obj);
}

#else

void lv_example_style_16(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_set_width(label, LV_PCT(80));
    lv_label_set_text(label, "LV_USE_DRAW_SW_COMPLEX_GRADIENTS is not␣
↪enabled");
    lv_label_set_long_mode(label, LV_LABEL_LONG_MODE_SCROLL_CIRCULAR);
    lv_obj_center(label);
}

#endif /*LV_USE_DRAW_SW_COMPLEX_GRADIENTS*/
```

(continues on next page)

(continua dalla pagina precedente)

```
#endif /*LV_BUILD_EXAMPLES*/
```

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

#if LV_USE_DRAW_SW_COMPLEX_GRADIENTS

/**
 * Using radial gradient as background
 */
void lv_example_style_17(void)
{
    static const lv_color_t grad_colors[2] = {
        LV_COLOR_MAKE(0x9B, 0x18, 0x42),
        LV_COLOR_MAKE(0x00, 0x00, 0x00),
    };

    int32_t width = lv_display_get_horizontal_resolution(NULL);
    int32_t height = lv_display_get_vertical_resolution(NULL);

    static lv_style_t style;
    lv_style_init(&style);

    /*First define a color gradient. In this example we use a purple to␣
↪black color map.*/
    static lv_grad_dsc_t grad;

    lv_grad_init_stops(&grad, grad_colors, NULL, NULL, sizeof(grad_colors)␣
↪/ sizeof(lv_color_t));

    /*Make a radial gradient with the center in the middle of the object,␣
↪extending to the farthest corner*/
    lv_grad_radial_init(&grad, LV_GRAD_CENTER, LV_GRAD_CENTER, LV_GRAD_
↪RIGHT, LV_GRAD_BOTTOM, LV_GRAD_EXTEND_PAD);

    /*Set gradient as background*/
    lv_style_set_bg_grad(&style, &grad);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_set_size(obj, width, height);
    lv_obj_center(obj);
}

#else

void lv_example_style_17(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_set_width(label, LV_PCT(80));
    lv_label_set_text(label, "LV_USE_DRAW_SW_COMPLEX_GRADIENTS is not␣
↪enabled");
    lv_label_set_long_mode(label, LV_LABEL_LONG_MODE_SCROLL_CIRCULAR);
    lv_obj_center(label);
}
```

(continues on next page)

(continua dalla pagina precedente)

```
#endif /*LV_USE_DRAW_SW_COMPLEX_GRADIENTS*/

#endif /*LV_BUILD_EXAMPLES*/
```

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

#if LV_USE_DRAW_SW_COMPLEX_GRADIENTS

/**
 * Using various gradients for button background
 */
void lv_example_style_18(void)
{
    static const lv_color_t grad_colors[2] = {
        LV_COLOR_MAKE(0x26, 0xa0, 0xda),
        LV_COLOR_MAKE(0x31, 0x47, 0x55),
    };

    /*Create a linear gradient going from the top left corner to the␣
↪bottom at an angle, with reflected color map*/
    static lv_style_t style_with_linear_gradient_bg;
    static lv_grad_dsc_t linear_gradient_dsc;     /*NOTE: the gradient␣
↪descriptor must be static or global variable!*/

    lv_style_init(&style_with_linear_gradient_bg);
    lv_grad_init_stops(&linear_gradient_dsc, grad_colors, NULL, NULL,␣
↪sizeof(grad_colors) / sizeof(lv_color_t));
    lv_grad_linear_init(&linear_gradient_dsc, lv_pct(0), lv_pct(0), lv_
↪pct(20), lv_pct(100), LV_GRAD_EXTEND_REFLECT);
    lv_style_set_bg_grad(&style_with_linear_gradient_bg, &linear_gradient_
↪dsc);
    lv_style_set_bg_opa(&style_with_linear_gradient_bg, LV_OPA_COVER);

    /*Create a radial gradient with the center in the top left 1/3rd of␣
↪the object, extending to the bottom right corner, with reflected color␣
↪map*/
    static lv_style_t style_with_radial_gradient_bg;
    static lv_grad_dsc_t radial_gradient_dsc;     /*NOTE: the gradient␣
↪descriptor must be static or global variable!*/

    lv_style_init(&style_with_radial_gradient_bg);
    lv_grad_init_stops(&radial_gradient_dsc, grad_colors, NULL, NULL,␣
↪sizeof(grad_colors) / sizeof(lv_color_t));
    lv_grad_radial_init(&radial_gradient_dsc, lv_pct(30), lv_pct(30), lv_
↪pct(100), lv_pct(100), LV_GRAD_EXTEND_REFLECT);
    lv_style_set_bg_grad(&style_with_radial_gradient_bg, &radial_gradient_
↪dsc);
    lv_style_set_bg_opa(&style_with_radial_gradient_bg, LV_OPA_COVER);

    /*Create buttons with different gradient styles*/

    lv_obj_t * btn;
    lv_obj_t * label;
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Simple horizontal gradient*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_set_style_bg_color(btn, grad_colors[0], 0);
    lv_obj_set_style_bg_grad_color(btn, grad_colors[1], 0);
    lv_obj_set_style_bg_grad_dir(btn, LV_GRAD_DIR_HOR, 0);
    lv_obj_set_size(btn, 150, 50);
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, -100);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Horizontal");
    lv_obj_center(label);

    /*Simple vertical gradient*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_set_style_bg_color(btn, grad_colors[0], 0);
    lv_obj_set_style_bg_grad_color(btn, grad_colors[1], 0);
    lv_obj_set_style_bg_grad_dir(btn, LV_GRAD_DIR_VER, 0);
    lv_obj_set_size(btn, 150, 50);
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, -40);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Vertical");
    lv_obj_center(label);

    /*Complex linear gradient*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_add_style(btn, &style_with_linear_gradient_bg, 0);
    lv_obj_set_size(btn, 150, 50);
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, 20);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Linear");
    lv_obj_center(label);

    /*Complex radial gradient*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_add_style(btn, &style_with_radial_gradient_bg, 0);
    lv_obj_set_size(btn, 150, 50);
    lv_obj_align(btn, LV_ALIGN_CENTER, 0, 80);

    label = lv_label_create(btn);
    lv_label_set_text(label, "Radial");
    lv_obj_center(label);
}

#else

void lv_example_style_18(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_set_width(label, LV_PCT(80));
    lv_label_set_text(label, "LV_USE_DRAW_SW_COMPLEX_GRADIENTS is not␣
↪enabled");
    lv_label_set_long_mode(label, LV_LABEL_LONG_MODE_SCROLL_CIRCULAR);
    lv_obj_center(label);
}
```

(continues on next page)

(continua dalla pagina precedente)

```
#endif /*LV_USE_DRAW_SW_COMPLEX_GRADIENTS*/

#endif /*LV_BUILD_EXAMPLES*/
```

For each gradient type, multiple color and opacity values can be assigned. These are called "stops". The maximum number of stops is limited to *LV_GRADIENT_MAX_STOPS*.

A gradient is basically a transition of colors and opacities between stops.

Besides just setting the color and opacity of each stop, it is also possible to set where they start relative to the whole gradient area.

For example with 3 stops it can be set like this:

- 10% red: 0--10% fully red
- 60% green: 10--60% transition from red to green, 60% is fully green
- 65% blue: fast transition from green to blue between 60%--65%. After 65% fully blue.

The position of the stops are called fractions or offsets and are 8 bit values where 0 is 0% and 255 is 100% of the whole gradient area.

lv_grad_init_stops(grad_dsc, colors, opas, fracs, cnt) initializes a gradient descriptor with stops containing the color, opacity and fraction of each stop.

```
static const lv_color_t colors[2] = {
    LV_COLOR_MAKE(0xe8, 0xe8, 0xe8),
    LV_COLOR_MAKE(0x79, 0x79, 0x79),
};

static const lv_opa_t opas[2] = {
    170,
    255,
};

static const uint8_t fracs[2] = {
    170,
    255,
};

lv_grad_init_stops(&grad, colors, opas, fracs, sizeof(colors) / sizeof(lv_
↪color_t));
```

If the opacity array is NULL 255 will be used for each stop. If the fractions array is NULL the colors will be distributed evenly. For example with 3 colors: 0%, 50%, 100%

Padding

Linear, radial, and conic gradients are defined between two points or angles. You can define how to pad the areas outside of the start and end points or angles:

- *LV_GRAD_EXTEND_PAD*: Repeat the same color
- *LV_GRAD_EXTEND_REPEAT*: Repeat the pattern
- *LV_GRAD_EXTEND_REFLECT*: Repeat the pattern normally and mirrored alternately

Horizontal and Vertical Gradients

The simplest and usually fastest gradient types are horizontal and vertical gradients.

After initializing the stops with *lv_grad_init_stops* call either *lv_grad_horizontal_init*(&grad_dsc) or *lv_grad_vertical_init*(&grad_dsc) to get a horizontal or vertical gradient descriptor.

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_LABEL

static void position_bullet(lv_event_t * e, lv_point_t * p)
{
    lv_indev_t * indev = (lv_indev_t *) lv_event_get_param(e);
    lv_indev_get_point(indev, p);

    lv_obj_t * bullet = lv_event_get_target_obj(e);
    lv_obj_t * parent = lv_obj_get_parent(bullet);

    p->x -= lv_obj_get_x(parent);
    p->y -= lv_obj_get_y(parent);

    int32_t w = lv_obj_get_width(parent);
    int32_t h = lv_obj_get_height(parent);
    lv_obj_set_pos(bullet, LV_CLAMP(5, p->x, w - 20), LV_CLAMP(5, p->y, h -
↪ 20));
}

static void frac_1_event_cb(lv_event_t * e)
{
    lv_style_t * style = (lv_style_t *) lv_event_get_user_data(e);
    lv_style_value_t v;
    if(lv_style_get_prop(style, LV_STYLE_BG_GRAD, &v) != LV_STYLE_RES_
↪FOUND) {
        LV_LOG_WARN("style prop not found");
    }
    else {
        lv_grad_dsc_t * dsc = (lv_grad_dsc_t *)v.ptr;

        lv_point_t p;
        position_bullet(e, &p);

        lv_obj_t * bullet = lv_event_get_target_obj(e);
        lv_obj_t * parent = lv_obj_get_parent(bullet);
        dsc->stops[0].frac = (uint8_t)LV_CLAMP(0, p.x * 255 / lv_obj_get_
↪width(parent), 255);

        lv_obj_invalidate(parent);
    }
}

static void frac_2_event_cb(lv_event_t * e)
{
    lv_style_t * style = (lv_style_t *) lv_event_get_user_data(e);
    lv_style_value_t v;
    if(lv_style_get_prop(style, LV_STYLE_BG_GRAD, &v) != LV_STYLE_RES_
↪FOUND) {
        LV_LOG_WARN("style prop not found");
```

(continues on next page)

(continua dalla pagina precedente)

```
    }
    else {
        lv_grad_dsc_t * dsc = (lv_grad_dsc_t *)v.ptr;

        lv_point_t p;
        position_bullet(e, &p);

        lv_obj_t * bullet = lv_event_get_target_obj(e);
        lv_obj_t * parent = lv_obj_get_parent(bullet);

        dsc->stops[1].frac = (uint8_t)LV_CLAMP(0, p.x * 255 / lv_obj_get_
↪width(parent), 255);
        lv_obj_invalidate(parent);
    }
}

/**
 * Play with a simple horizontal gradient.
 * Adjust the stop positions of the gradient.
 */
void lv_example_grad_1(void)
{
    static const lv_color_t grad_colors[2] = {
        LV_COLOR_MAKE(0xff, 0x00, 0x00),
        LV_COLOR_MAKE(0x00, 0xff, 0x00),
    };

    static const lv_opa_t grad_opa[2] = {
        LV_OPA_100,
        LV_OPA_0,
    };

    static const uint8_t frac[2] = {
        20 * 255 / 100, /*20%*/
        80 * 255 / 100, /*80%*/
    };

    static lv_style_t style;
    lv_style_init(&style);

    static lv_grad_dsc_t grad_dsc;
    lv_grad_init_stops(&grad_dsc, grad_colors, grad_opa, frac, sizeof(grad_
↪colors) / sizeof(lv_color_t));
    lv_grad_horizontal_init(&grad_dsc);

    /*Set gradient as background*/
    lv_style_set_bg_grad(&style, &grad_dsc);
    lv_style_set_border_width(&style, 2);
    lv_style_set_pad_all(&style, 0);
    lv_style_set_radius(&style, 12);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_set_size(obj, lv_pct(80), lv_pct(80));
    lv_obj_center(obj);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_t * frac_1 = lv_button_create(obj);
    lv_obj_set_size(frac_1, 15, 15);
    lv_obj_set_style_bg_color(frac_1, lv_color_hex(0xff00ff), 0);
    lv_obj_add_event_cb(frac_1, frac_1_event_cb, LV_EVENT_PRESSING, &
↪style);
    lv_obj_set_ext_click_area(frac_1, 5);
    lv_obj_set_pos(frac_1, lv_pct(20), lv_pct(50));

    lv_obj_t * frac_2 = lv_button_create(obj);
    lv_obj_set_size(frac_2, 15, 15);
    lv_obj_set_style_bg_color(frac_2, lv_color_hex(0xffff00), 0);
    lv_obj_add_event_cb(frac_2, frac_2_event_cb, LV_EVENT_PRESSING, &
↪style);
    lv_obj_set_ext_click_area(frac_2, 5);
    lv_obj_set_pos(frac_2, lv_pct(80), lv_pct(50));
}

#endif
```

Linear Gradients

The linear (or skew) gradient are similar to horizontal or vertical gradient but the angle of the gradient can be controlled.

The linear gradient will be rendered along a line defined by 2 points.

After initializing the stops with *lv_grad_init_stops()* call *lv_grad_linear_init*(`&grad_dsc, from_x, from_y, to_x, to_y, LV_GRAD_EXTEND_...`) with your point values and extend pattern strategy to get a linear gradient descriptor.

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_LABEL
#if LV_USE_DRAW_SW_COMPLEX_GRADIENTS
static void position_bullet(lv_event_t * e, lv_point_t * p)
{
    lv_indev_t * indev = (lv_indev_t *) lv_event_get_param(e);
    lv_indev_get_point(indev, p);

    lv_obj_t * bullet = lv_event_get_target_obj(e);
    lv_obj_t * parent = lv_obj_get_parent(bullet);

    p->x -= lv_obj_get_x(parent);
    p->y -= lv_obj_get_y(parent);

    int32_t w = lv_obj_get_width(parent);
    int32_t h = lv_obj_get_height(parent);
    lv_obj_set_pos(bullet, LV_CLAMP(5, p->x, w - 20), LV_CLAMP(5, p->y, h -
↪ 20));
}

static void start_event_cb(lv_event_t * e)
{
    lv_style_t * style = (lv_style_t *) lv_event_get_user_data(e);
    lv_style_value_t v;
    lv_style_get_prop(style, LV_STYLE_BG_GRAD, &v);
    lv_grad_dsc_t * dsc = (lv_grad_dsc_t *)v.ptr;
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_point_t p;
    position_bullet(e, &p);

    dsc->params.linear.start.x = p.x;
    dsc->params.linear.start.y = p.y;

    lv_obj_t * bullet = lv_event_get_target_obj(e);
    lv_obj_t * parent = lv_obj_get_parent(bullet);
    lv_obj_invalidate(parent);
}

static void end_event_cb(lv_event_t * e)
{
    lv_style_t * style = (lv_style_t *) lv_event_get_user_data(e);
    lv_style_value_t v;
    lv_style_get_prop(style, LV_STYLE_BG_GRAD, &v);
    lv_grad_dsc_t * dsc = (lv_grad_dsc_t *)v.ptr;

    lv_point_t p;
    position_bullet(e, &p);

    dsc->params.linear.end.x = p.x;
    dsc->params.linear.end.y = p.y;

    lv_obj_t * bullet = lv_event_get_target_obj(e);
    lv_obj_t * parent = lv_obj_get_parent(bullet);
    lv_obj_invalidate(parent);
}

/**
 * Play with the linear gradient.
 * Adjust the 2 point in between the a linear gradient can be drawn (can␣
↪be skew as well)
 */
void lv_example_grad_2(void)
{
    static const lv_color_t grad_colors[2] = {
        LV_COLOR_MAKE(0xff, 0x00, 0x00),
        LV_COLOR_MAKE(0x00, 0xff, 0x00),
    };

    static const lv_opa_t grad_opa[2] = {
        LV_OPA_100,
        LV_OPA_0,
    };


    static lv_style_t style;
    lv_style_init(&style);

    /*First define a color gradient. In this example we use a purple to␣
↪black color map.*/
    static lv_grad_dsc_t grad;

    lv_grad_init_stops(&grad, grad_colors, grad_opa, NULL, sizeof(grad_
```

(continues on next page)

(continua dalla pagina precedente)

```
↪colors) / sizeof(lv_color_t));

    lv_grad_linear_init(&grad, 100, 100, 200, 150, LV_GRAD_EXTEND_PAD);

    /*Set gradient as background*/
    lv_style_set_bg_grad(&style, &grad);
    lv_style_set_border_width(&style, 2);
    lv_style_set_pad_all(&style, 0);
    lv_style_set_radius(&style, 12);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_set_size(obj, lv_pct(80), lv_pct(80));
    lv_obj_center(obj);

    lv_obj_t * start = lv_button_create(obj);
    lv_obj_set_size(start, 15, 15);
    lv_obj_set_style_bg_color(start, lv_color_hex(0x0000ff), 0);
    lv_obj_add_event_cb(start, start_event_cb, LV_EVENT_PRESSING, &style);
    lv_obj_set_ext_click_area(start, 5);
    lv_obj_set_pos(start, 100, 100);

    lv_obj_t * end = lv_button_create(obj);
    lv_obj_set_size(end, 15, 15);
    lv_obj_set_style_bg_color(end, lv_color_hex(0x00ffff), 0);
    lv_obj_add_event_cb(end, end_event_cb, LV_EVENT_PRESSING, &style);
    lv_obj_set_ext_click_area(end, 5);
    lv_obj_set_pos(end, 200, 150);
}

#else

void lv_example_grad_2(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "LV_USE_DRAW_SW_COMPLEX_GRADIENTS needs to be␣
↪enabled");
    lv_obj_center(label);
}

#endif /*LV_USE_DRAW_SW_COMPLEX_GRADIENTS*/

#endif
```

Radial Gradients

The radial gradient is described by two circles: an outer circle and an inner circle (also called the focal point). The gradient will be calculated between the focal point's circle and the edge of the outer circle.

If the center of the focal point and the center of the main circle are the same, the gradient will spread evenly in all directions. If the center points are not the same, the gradient will have an egg shape.

The focal point's circle should be inside the main circle.

After initializing the stops with *lv_grad_init_stops()*, the outer circle can be set by: *lv_grad_radial_init*(&grad_dsc, center_x, center_y, edge_x, edge_y, LV_GRAD_EXTEND_...)

For both the center and edge coordinates, `px` or `lv_pct()` values can be used.

The inner circle (focal point) can be set with: *`lv_grad_radial_set_focal`*`(&grad_dsc, center_x, center_y, radius)`

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_LABEL

#if LV_USE_DRAW_SW_COMPLEX_GRADIENTS

static void position_bullet(lv_event_t * e, lv_point_t * p)
{
    lv_indev_t * indev = (lv_indev_t *) lv_event_get_param(e);
    lv_indev_get_point(indev, p);

    lv_obj_t * bullet = lv_event_get_target_obj(e);
    lv_obj_t * parent = lv_obj_get_parent(bullet);

    p->x -= lv_obj_get_x(parent);
    p->y -= lv_obj_get_y(parent);

    int32_t w = lv_obj_get_width(parent);
    int32_t h = lv_obj_get_height(parent);
    lv_obj_set_pos(bullet, LV_CLAMP(5, p->x, w - 20), LV_CLAMP(5, p->y, h -
↪ 20));
}

static void focal_event_cb(lv_event_t * e)
{
    lv_style_t * style = (lv_style_t *) lv_event_get_user_data(e);
    lv_style_value_t v;
    lv_style_get_prop(style, LV_STYLE_BG_GRAD, &v);
    lv_grad_dsc_t * dsc = (lv_grad_dsc_t *)v.ptr;

    lv_point_t p;
    position_bullet(e, &p);

    dsc->params.radial.focal.x = p.x;
    dsc->params.radial.focal.y = p.y;
    dsc->params.radial.focal_extent.x = p.x + 10;
    dsc->params.radial.focal_extent.y = p.y;

    lv_obj_t * bullet = lv_event_get_target_obj(e);
    lv_obj_t * parent = lv_obj_get_parent(bullet);
    lv_obj_invalidate(parent);
}

static void end_event_cb(lv_event_t * e)
{
    lv_style_t * style = (lv_style_t *) lv_event_get_user_data(e);
    lv_style_value_t v;
    lv_style_get_prop(style, LV_STYLE_BG_GRAD, &v);
    lv_grad_dsc_t * dsc = (lv_grad_dsc_t *)v.ptr;

    lv_point_t p;
    position_bullet(e, &p);

    dsc->params.radial.end.x = p.x;
```

(continues on next page)

(continua dalla pagina precedente)

```
    dsc->params.radial.end.y = p.y;
    dsc->params.radial.end_extent.x = p.x + 100;
    dsc->params.radial.end_extent.y = p.y;
    lv_obj_t * bullet = lv_event_get_target_obj(e);
    lv_obj_t * parent = lv_obj_get_parent(bullet);
    lv_obj_invalidate(parent);
}

/**
 * Play with the radial gradient
 * Adjust the end circle and focal point position.
 * The radius of the end circle and an focal point are hardcoded in the␣
↪example.
 */
void lv_example_grad_3(void)
{
    static const lv_color_t grad_colors[2] = {
        LV_COLOR_MAKE(0xff, 0x00, 0x00),
        LV_COLOR_MAKE(0x00, 0xff, 0x00),
    };

    static const lv_opa_t grad_opa[2] = {
        LV_OPA_100,
        LV_OPA_0,
    };

    static lv_style_t style;
    lv_style_init(&style);

    /*First define a color gradient. In this example we use a purple to␣
↪black color map.*/
    static lv_grad_dsc_t grad;

    lv_grad_init_stops(&grad, grad_colors, grad_opa, NULL, sizeof(grad_
↪colors) / sizeof(lv_color_t));

    /*Init a radial gradient where the center is at 100;100
     *and the edge of the circle is at 200;100.
     *Try LV_GRAD_EXTEND_REFLECT and LV_GRAD_EXTEND_REPEAT too. */
    lv_grad_radial_init(&grad, 100, 100, 200, 100, LV_GRAD_EXTEND_PAD);

    /*The gradient will be calculated between the focal point's circle and␣
↪the
     *edge of the circle. If the center of the focal point and the
     *center of the main circle is the same, the gradient will spread
     *evenly in all directions. The focal point should be inside the
     *main circle.*/
    lv_grad_radial_set_focal(&grad, 50, 50, 10);

    /*Set the widget containing the gradient*/
    lv_style_set_bg_grad(&style, &grad);
    lv_style_set_border_width(&style, 2);
    lv_style_set_pad_all(&style, 0);
    lv_style_set_radius(&style, 12);

    /*Create an object with the new style*/
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_set_size(obj, lv_pct(80), lv_pct(80));
    lv_obj_center(obj);

    lv_obj_t * focal = lv_button_create(obj);
    lv_obj_set_size(focal, 15, 15);
    lv_obj_set_style_bg_color(focal, lv_color_hex(0x0000ff), 0);
    lv_obj_add_event_cb(focal, focal_event_cb, LV_EVENT_PRESSING, &style);
    lv_obj_set_ext_click_area(focal, 5);
    lv_obj_set_pos(focal, 50, 50);

    lv_obj_t * end = lv_button_create(obj);
    lv_obj_set_size(end, 15, 15);
    lv_obj_set_style_bg_color(end, lv_color_hex(0x00ffff), 0);
    lv_obj_add_event_cb(end, end_event_cb, LV_EVENT_PRESSING, &style);
    lv_obj_set_ext_click_area(end, 5);
    lv_obj_set_pos(end, 100, 100);
}
#else

void lv_example_grad_3(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "LV_USE_DRAW_SW_COMPLEX_GRADIENTS needs to be␣
↪enabled");
    lv_obj_center(label);
}

#endif /*LV_USE_DRAW_SW_COMPLEX_GRADIENTS*/

#endif
```

Conic Gradients

The conic gradient is defined between the angles of a circle, and colors are mapped to each angle.

After initializing the stops with *lv_grad_init_stops()*, the conic gradient can be set up with: *lv_grad_conical_init*(&grad, center_x, center_y, angle_start, angle_end, LV_GRAD_EXTEND_...)

For both the center and edge coordinates, `px` or `lv_pct()` values can be used.

The zero angle is on the right-hand side, and 90 degrees is at the bottom.

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_LABEL

#if LV_USE_DRAW_SW_COMPLEX_GRADIENTS

static void position_bullet(lv_event_t * e, lv_point_t * p)
{
    lv_indev_t * indev = (lv_indev_t *) lv_event_get_param(e);
    lv_indev_get_point(indev, p);

    lv_obj_t * bullet = lv_event_get_target_obj(e);
    lv_obj_t * parent = lv_obj_get_parent(bullet);
```

(continues on next page)

(continua dalla pagina precedente)

```
    p->x -= lv_obj_get_x(parent);
    p->y -= lv_obj_get_y(parent);

    int32_t w = lv_obj_get_width(parent);
    int32_t h = lv_obj_get_height(parent);
    lv_obj_set_pos(bullet, LV_CLAMP(5, p->x, w - 20), LV_CLAMP(5, p->y, h -
↪ 20));
}

static void start_event_cb(lv_event_t * e)
{
    lv_style_t * style = (lv_style_t *) lv_event_get_user_data(e);
    lv_style_value_t v;
    lv_style_get_prop(style, LV_STYLE_BG_GRAD, &v);
    lv_grad_dsc_t * dsc = (lv_grad_dsc_t *)v.ptr;

    lv_point_t p;
    position_bullet(e, &p);

    lv_obj_t * bullet = lv_event_get_target_obj(e);
    lv_obj_t * parent = lv_obj_get_parent(bullet);
    p.x -= lv_obj_get_width(parent) / 2;
    p.y -= lv_obj_get_height(parent) / 2;

    dsc->params.conical.start_angle = lv_atan2(p.y, p.x);
    lv_obj_invalidate(parent);
}

static void end_event_cb(lv_event_t * e)
{
    lv_style_t * style = (lv_style_t *) lv_event_get_user_data(e);
    lv_style_value_t v;
    lv_style_get_prop(style, LV_STYLE_BG_GRAD, &v);
    lv_grad_dsc_t * dsc = (lv_grad_dsc_t *)v.ptr;

    lv_point_t p;
    position_bullet(e, &p);

    lv_obj_t * bullet = lv_event_get_target_obj(e);
    lv_obj_t * parent = lv_obj_get_parent(bullet);
    p.x -= lv_obj_get_width(parent) / 2;
    p.y -= lv_obj_get_height(parent) / 2;

    dsc->params.conical.end_angle = lv_atan2(p.y, p.x);
    lv_obj_invalidate(parent);
}

/**
 * Play with the conical gradient
 */
void lv_example_grad_4(void)
{
    static const lv_color_t grad_colors[2] = {
        LV_COLOR_MAKE(0xff, 0x00, 0x00),
        LV_COLOR_MAKE(0x00, 0xff, 0x00),
```

(continues on next page)

(continua dalla pagina precedente)

```
    };

    static const lv_opa_t grad_opa[2] = {
        LV_OPA_100,
        LV_OPA_0,
    };

    static lv_style_t style;
    lv_style_init(&style);

    /*First define a color gradient. In this example we use a purple to␣
↪black color map.*/
    static lv_grad_dsc_t grad;

    lv_grad_init_stops(&grad, grad_colors, grad_opa, NULL, sizeof(grad_
↪colors) / sizeof(lv_color_t));

    lv_grad_conical_init(&grad, lv_pct(50), lv_pct(50), 0, 180, LV_GRAD_
↪EXTEND_PAD);

    /*Set gradient as background*/
    lv_style_set_bg_grad(&style, &grad);
    lv_style_set_border_width(&style, 2);
    lv_style_set_pad_all(&style, 0);
    lv_style_set_radius(&style, 12);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_set_size(obj, lv_pct(80), lv_pct(80));
    lv_obj_center(obj);

    lv_obj_t * start = lv_button_create(obj);
    lv_obj_set_size(start, 15, 15);
    lv_obj_set_style_bg_color(start, lv_color_hex(0x0000ff), 0);
    lv_obj_add_event_cb(start, start_event_cb, LV_EVENT_PRESSING, &style);
    lv_obj_set_ext_click_area(start, 5);
    lv_obj_set_pos(start, lv_pct(80), lv_pct(50));

    lv_obj_t * end = lv_button_create(obj);
    lv_obj_set_size(end, 15, 15);
    lv_obj_set_style_bg_color(end, lv_color_hex(0x00ffff), 0);
    lv_obj_add_event_cb(end, end_event_cb, LV_EVENT_PRESSING, &style);
    lv_obj_set_ext_click_area(end, 5);
    lv_obj_set_pos(end, lv_pct(20), lv_pct(50));
}

#else

void lv_example_grad_4(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "LV_USE_DRAW_SW_COMPLEX_GRADIENTS needs to be␣
↪enabled");
    lv_obj_center(label);
}
```

(continues on next page)

(continua dalla pagina precedente)

```
#endif /*LV_USE_DRAW_SW_COMPLEX_GRADIENTS*/

#endif
```

Border Draw Descriptor

The `lv_draw_border_dsc_t` border descriptor has radius, opacity, width, color, and side fields. If the opacity or width is 0, no Draw Task will be created.

`side` can contain ORed values of `lv_border_side_t`, such as `LV_BORDER_SIDE_BOTTOM`. `LV_BORDER_SIDE_ALL` applies to all sides, while `LV_BORDER_SIDE_INTERNAL` is used by higher layers (e.g. a table Widget) to calculate border sides. However, the drawing routine receives only simpler values.

The following functions are used for border drawing:

- `lv_draw_border_dsc_init(&dsc)` initializes a border Draw Task.
- `lv_draw_sw_border(layer, &dsc, area)` creates a Draw Task to draw a border inward from its area.
- `lv_draw_task_get_border_dsc(draw_task)` retrieves the border descriptor from a Draw Task.

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

/**
 * Using the border style properties
 */
void lv_example_style_3(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    /*Set a background color and a radius*/
    lv_style_set_radius(&style, 10);
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_bg_color(&style, lv_palette_lighten(LV_PALETTE_GREY, 1));

    /*Add border to the bottom+right*/
    lv_style_set_border_color(&style, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_border_width(&style, 5);
    lv_style_set_border_opa(&style, LV_OPA_50);
    lv_style_set_border_side(&style, (lv_border_side_t)(LV_BORDER_SIDE_
↪BOTTOM | LV_BORDER_SIDE_RIGHT));

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_center(obj);
}

#endif
```

Outlines

The outline is similar to the border but is drawn outside the object's draw area.

In practice, there is no dedicated outline descriptor like `lv_draw_outline_dsc_t`, because from the rendering perspective, the outline is simply another border rendered outside the object's bounds.

The outline is used only in *`lv_draw_rect_dsc_t`* for convenience. The two differences compared to borders in *`lv_draw_rect_dsc_t`* are:

- There is an `outline_pad` property to specify the gap between the target area and the inner side of the outline. It can be negative. For example, if `outline_pad = -width`, the outline will resemble a border.
- There is no `border_side` property for the outline. It's always rendered as a full rectangle.

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

/**
 * Using the outline style properties
 */
void lv_example_style_4(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    /*Set a background color and a radius*/
    lv_style_set_radius(&style, 5);
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_bg_color(&style, lv_palette_lighten(LV_PALETTE_GREY, 1));

    /*Add outline*/
    lv_style_set_outline_width(&style, 2);
    lv_style_set_outline_color(&style, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_outline_pad(&style, 8);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_center(obj);
}

#endif
```

Box Shadow Draw Descriptor

The *`lv_draw_box_shadow_dsc_t`* box shadow descriptor describes a **rounded rectangle-shaped shadow**. It cannot generate shadows for arbitrary shapes, text, or images. It includes the following fields:

radius
: Radius, `LV_RADIUS_CIRCLE`.

color
: Shadow color.

width
: Shadow width (blur radius).

spread
: Expands the rectangle in all directions; can be negative.

ofs_x
: Horizontal offset.

ofs_y
Vertical offset.

opa
Opacity (0--255 range). Values like `LV_OPA_TRANSP`, `LV_OPA_10`, etc., can also be used.

bg_cover
Set to 1 if the background will cover the shadow (a hint for the renderer to skip masking).

Note: Rendering large shadows may be slow or memory-intensive.

The following functions are used for box shadow drawing:

- *`lv_draw_box_shadow_dsc_init`*`(&dsc)` initializes a box shadow Draw Task.
- *`lv_draw_sw_box_shadow`*`(layer, &dsc, area)` creates a Draw Task for a rectangle's shadow. The shadow's size and position depend on the width, spread, and offset.
- *`lv_draw_task_get_box_shadow_dsc`*`(draw_task)` retrieves the box shadow descriptor from a Draw Task.

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

/**
 * Using the Shadow style properties
 */
void lv_example_style_5(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    /*Set a background color and a radius*/
    lv_style_set_radius(&style, 5);
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_bg_color(&style, lv_palette_lighten(LV_PALETTE_GREY, 1));

    /*Add a shadow*/
    lv_style_set_shadow_width(&style, 55);
    lv_style_set_shadow_color(&style, lv_palette_main(LV_PALETTE_BLUE));

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_obj_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_center(obj);
}

#endif
```

Image Draw Descriptor

The `lv_draw_image_dsc_t` image descriptor defines the parameters for image drawing. It is a complex descriptor with the following options:

src
The image source, either a pointer to `lv_image_dsc_t` or a file path.

opa
Opacity in the 0--255 range. Options like `LV_OPA_TRANSP`, `LV_OPA_10`, etc., can also be used.

clip_radius
Clips the corners of the image with this radius. Use `LV_RADIUS_CIRCLE` for the maximum radius.

rotation
Image rotation in 0.1-degree units (e.g., 234 means 23.4°).

scale_x
Horizontal scaling (zoom) of the image. 256 (LV_SCALE_NONE) means no zoom, 512 doubles the size, and 128 halves it.

scale_y
Same as `scale_x` but for vertical scaling.

skew_x
Horizontal skew (parallelogram-like transformation) in 0.1-degree units (e.g., 456 means 45.6°).

skew_y
Vertical skew, similar to `skew_x`.

pivot
The pivot point for transformations (scaling and rotation). (0,0) is the top-left corner of the image and can be set outside the image.

bitmap_mask_src
Pointer to an A8 or L8 image descriptor used to mask the image. The mask is always center-aligned.

recolor
Mixes this color with the image. For *`LV_COLOR_FORMAT_A8`*, this will be the visible pixels' color.

recolor_opa
Intensity of recoloring (0 means no recoloring, 255 means full cover).

blend_mode
Defines how to blend image pixels with the background. See *`lv_blend_mode_t`* for more details.

antialias
Set to 1 to enable anti-aliasing for transformations.

tile
Tiles the image (repeats it both horizontally and vertically) if the image is smaller than the `image_area` field in `lv_draw_image_dsc_t`.

image_area
Indicates the original, non-clipped area where the image is drawn. This is essential for:

1. Layer rendering, where only part of a layer may be rendered and `clip_radius` needs the original image dimensions.
2. Tiling, where the draw area is larger than the image.

sup
Internal field to store information about the palette or color of A8 images.

Functions for image drawing:

- *`lv_draw_image_dsc_init`*`(&dsc)` initializes an image draw descriptor.
- *`lv_draw_image`*`(layer, &dsc, area)` creates a task to draw an image in a given area.
- *`lv_draw_task_get_image_dsc`*`(draw_task)` retrieves the image descriptor from a task.

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES
```

(continues on next page)

(continua dalla pagina precedente)

```
#define CANVAS_WIDTH  50
#define CANVAS_HEIGHT  50

/**
 * Draw an image to the canvas
 */
void lv_example_canvas_6(void)
{
    /*Create a buffer for the canvas*/
    static uint8_t cbuf[LV_CANVAS_BUF_SIZE(CANVAS_WIDTH, CANVAS_HEIGHT, 32,
↪ LV_DRAW_BUF_STRIDE_ALIGN)];

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_buffer(canvas, cbuf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    LV_IMAGE_DECLARE(img_star);
    lv_draw_image_dsc_t dsc;
    lv_draw_image_dsc_init(&dsc);
    dsc.src = &img_star;

    lv_area_t coords = {10, 10, 10 + img_star.header.w - 1, 10 + img_star.
↪header.h - 1};

    lv_draw_image(&layer, &dsc, &coords);

    lv_canvas_finish_layer(canvas, &layer);
}
#endif
```

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_IMAGE

/**
 * Using the Image style properties
 */
void lv_example_style_6(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    /*Set a background color and a radius*/
    lv_style_set_radius(&style, 5);
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_bg_color(&style, lv_palette_lighten(LV_PALETTE_GREY, 3));
    lv_style_set_border_width(&style, 2);
    lv_style_set_border_color(&style, lv_palette_main(LV_PALETTE_BLUE));

    lv_style_set_image_recolor(&style, lv_palette_main(LV_PALETTE_BLUE));
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_set_image_recolor_opa(&style, LV_OPA_50);
    lv_style_set_transform_rotation(&style, 300);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_image_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);

    LV_IMAGE_DECLARE(img_cogwheel_argb);
    lv_image_set_src(obj, &img_cogwheel_argb);

    lv_obj_center(obj);
}

#endif
```

Layers - Special Images

Layers are treated as images, so an `lv_draw_image_dsc_t` can describe how layers are blended into their parent layers. All image features apply to layers as well.

`lv_draw_layer(layer, &dsc, area)` initializes the blending of a layer back to its parent layer. Additionally, image-drawing-related functions can be used for layers.

For more details, see *I Layer*.

Label Draw Descriptor

The *`lv_draw_label_dsc_t`* label descriptor provides extensive options for controlling text rendering:

text
: The text to render.

font
: Font to use, with support for fallback fonts.

color
: Text color.

opa
: Text opacity.

line_space
: Additional space between lines.

letter_space
: Additional space between characters.

ofs_x
: Horizontal text offset.

ofs_y
: Vertical text offset.

sel_start
: Index of the first character for selection (not byte index). `LV_DRAW_LABEL_NO_TXT_SEL` means no selection.

sel_end
: Index of the last character for selection.

sel_color
: Color of selected characters.

sel_bg_color
Background color for selected characters.

align
Text alignment. See `lv_text_align_t`.

bidi_dir
Base direction for right-to-left text rendering (e.g., Arabic). See `lv_base_dir_t`.

decor
Text decoration, e.g., underline. See `lv_text_decor_t`.

flag
Flags for text rendering. See `lv_text_flag_t`.

text_length
Number of characters to render (0 means render until `0`).

text_local
Set to 1 to allocate a buffer and copy the text.

text_static
Indicates `text` is constant and its pointer can be cached.

hint
Pointer to externally stored data to speed up rendering. See `lv_draw_label_hint_t`.

Functions for text drawing:

- `lv_draw_label_dsc_init(&dsc)` initializes a label draw descriptor.
- `lv_draw_label(layer, &dsc, area)` creates a task to render text in an area.
- `lv_draw_character(layer, &dsc, point, unicode_letter)` creates a task to draw a character at a specific point.
- `lv_draw_task_get_label_dsc(draw_task)` retrieves the label descriptor from a task.

For character-specific drawing in draw units, use `lv_draw_label_iterate_characters(draw_unit, draw_dsc, area, callback)`. This iterates through all characters, calculates their positions, and calls the callback for rendering each character. For callback details, see `lv_draw_glyph_cb_t`.

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_FONT_MONTSERRAT_18 && LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  50
#define CANVAS_HEIGHT  50

/**
 * Draw a text to the canvas
 */
void lv_example_canvas_4(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf);
    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_draw_label_dsc_t dsc;
    lv_draw_label_dsc_init(&dsc);
    dsc.color = lv_palette_main(LV_PALETTE_RED);
    dsc.font = &lv_font_montserrat_18;
    dsc.decor = LV_TEXT_DECOR_UNDERLINE;
    dsc.text = "Hello";

    lv_area_t coords = {10, 10, 30, 60};

    lv_draw_label(&layer, &dsc, &coords);

    lv_canvas_finish_layer(canvas, &layer);

}
#endif
```

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_LABEL

/**
 * Using the text style properties
 */
void lv_example_style_8(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    lv_style_set_radius(&style, 5);
    lv_style_set_bg_opa(&style, LV_OPA_COVER);
    lv_style_set_bg_color(&style, lv_palette_lighten(LV_PALETTE_GREY, 2));
    lv_style_set_border_width(&style, 2);
    lv_style_set_border_color(&style, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_pad_all(&style, 10);

    lv_style_set_text_color(&style, lv_palette_main(LV_PALETTE_BLUE));
    lv_style_set_text_letter_space(&style, 5);
    lv_style_set_text_line_space(&style, 20);
    lv_style_set_text_decor(&style, LV_TEXT_DECOR_UNDERLINE);

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

    lv_obj_center(obj);
}

#endif
```

Arc Draw Descriptor

The `lv_draw_arc_dsc_t` arc descriptor defines arc rendering with these fields:

color
Arc color.

img_src
Image source for the arc, or `NULL` if unused.

width
Arc thickness.

start_angle
Starting angle in degrees (e.g., 0° is 3 o'clock, 90° is 6 o'clock).

end_angle
Ending angle.

center
Arc center point.

radius
Arc radius.

opa
Arc opacity (0--255).

rounded
Rounds the arc ends.

Functions for arc drawing:

- `lv_draw_arc_dsc_init(&dsc)` initializes an arc descriptor.
- `lv_draw_arc(layer, &dsc)` creates a task to render an arc.
- `lv_draw_task_get_arc_dsc(draw_task)` retrieves arc descriptor from task.

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  50
#define CANVAS_HEIGHT  50

/**
 * Draw an arc to the canvas
 */
void lv_example_canvas_5(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf);
    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_draw_arc_dsc_t dsc;
    lv_draw_arc_dsc_init(&dsc);
    dsc.color = lv_palette_main(LV_PALETTE_RED);
    dsc.width = 5;
    dsc.center.x = 25;
    dsc.center.y = 25;
    dsc.width = 10;
    dsc.radius = 15;
    dsc.start_angle = 0;
    dsc.end_angle = 220;

    lv_draw_arc(&layer, &dsc);

    lv_canvas_finish_layer(canvas, &layer);

}
#endif
```

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

/**
 * Using the Arc style properties
 */
void lv_example_style_7(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    lv_style_set_arc_color(&style, lv_palette_main(LV_PALETTE_RED));
    lv_style_set_arc_width(&style, 4);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_arc_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);
    lv_obj_center(obj);
}
#endif
```

Line Draw Descriptor

The `lv_draw_line_dsc_t` line descriptor defines line rendering with these fields:

p1
First point of line (supports floating-point coordinates).

p2
Second point of line (supports floating-point coordinates).

color
Line color.

width
Line thickness.

opa
Line opacity (0--255).

dash_width
Length of dashes (0 means no dashes).

dash_gap
Length of gaps between dashes (0 means no dashes).

round_start
Rounds the line start.

round_end
Rounds the line end.

raw_end
Set to 1 to skip end calculations if they are unnecessary.

Functions for line drawing:

- `lv_draw_line_dsc_init(&dsc)` initializes a line descriptor.
- `lv_draw_line(layer, &dsc)` creates a task to draw a line.
- `lv_draw_task_get_line_dsc(draw_task)` retrieves line descriptor.

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS&& LV_BUILD_EXAMPLES

#define CANVAS_WIDTH  50
#define CANVAS_HEIGHT  50

/**
 * Draw a line to the canvas
 */
void lv_example_canvas_7(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf);
    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_draw_line_dsc_t dsc;
    lv_draw_line_dsc_init(&dsc);
    dsc.color = lv_palette_main(LV_PALETTE_RED);
    dsc.width = 4;
    dsc.round_end = 1;
    dsc.round_start = 1;
    dsc.p1.x = 15;
    dsc.p1.y = 15;
    dsc.p2.x = 35;
    dsc.p2.y = 10;
    lv_draw_line(&layer, &dsc);

    lv_canvas_finish_layer(canvas, &layer);
```

(continues on next page)

(continua dalla pagina precedente)

```
}
#endif
```

```
#include "../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_LINE

/**
 * Using the line style properties
 */
void lv_example_style_9(void)
{
    static lv_style_t style;
    lv_style_init(&style);

    lv_style_set_line_color(&style, lv_palette_main(LV_PALETTE_GREY));
    lv_style_set_line_width(&style, 6);
    lv_style_set_line_rounded(&style, true);

    /*Create an object with the new style*/
    lv_obj_t * obj = lv_line_create(lv_screen_active());
    lv_obj_add_style(obj, &style, 0);

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

    lv_obj_center(obj);
}

#endif
```

Triangle Draw Descriptor

Triangles are defined by `lv_draw_triangle_dsc_t`, which includes:

p[3]
3 points for the triangle's vertices.

color
Triangle color.

opa
Triangle opacity.

grad
Gradient options. If `grad.dir` is not `LV_GRAD_DIR_NONE`, the `color` field is ignored. The `opa` field adjusts overall opacity.

Functions for triangle drawing: - `lv_draw_triangle_dsc_init(&dsc)` initializes a triangle descriptor. - `lv_draw_triangle(layer, &dsc)` creates a task to draw a triangle. - `lv_draw_task_get_triangle_dsc(draw_task)` retrieves triangle descriptor.

```
#include "../../lv_examples.h"
#if LV_USE_CANVAS && LV_BUILD_EXAMPLES


#define CANVAS_WIDTH  150
#define CANVAS_HEIGHT 150
```

(continues on next page)

(continua dalla pagina precedente)

```
/**
 * Draw a triangle to the canvas
 */
void lv_example_canvas_9(void)
{
    /*Create a buffer for the canvas*/
    LV_DRAW_BUF_DEFINE_STATIC(draw_buf, CANVAS_WIDTH, CANVAS_HEIGHT, LV_
↪COLOR_FORMAT_ARGB8888);
    LV_DRAW_BUF_INIT_STATIC(draw_buf);

    /*Create a canvas and initialize its palette*/
    lv_obj_t * canvas = lv_canvas_create(lv_screen_active());
    lv_canvas_set_draw_buf(canvas, &draw_buf);
    lv_canvas_fill_bg(canvas, lv_color_hex3(0xccc), LV_OPA_COVER);
    lv_obj_center(canvas);

    lv_layer_t layer;
    lv_canvas_init_layer(canvas, &layer);

    lv_draw_triangle_dsc_t tri_dsc;
    lv_draw_triangle_dsc_init(&tri_dsc);
    tri_dsc.p[0].x = 10;
    tri_dsc.p[0].y = 10;
    tri_dsc.p[1].x = 100;
    tri_dsc.p[1].y = 30;
    tri_dsc.p[2].x = 50;
    tri_dsc.p[2].y = 100;

    tri_dsc.grad.stops_count = 2;
    tri_dsc.grad.dir = LV_GRAD_DIR_VER;
    tri_dsc.grad.stops[0].color = lv_color_hex(0xff0000);
    tri_dsc.grad.stops[0].frac = 64;    /*Start at 25%*/
    tri_dsc.grad.stops[0].opa = LV_OPA_COVER;
    tri_dsc.grad.stops[1].color = lv_color_hex(0x0000ff);
    tri_dsc.grad.stops[1].opa = LV_OPA_TRANSP;
    tri_dsc.grad.stops[1].frac = 3 * 64;    /*End at 75%*/

    tri_dsc.opa = 128;  /*Set the overall opacity to 50%*/

    lv_draw_triangle(&layer, &tri_dsc);

    lv_canvas_finish_layer(canvas, &layer);
}

#endif
```

Vector Draw Descriptor

TODO

Masking Operation

There are several options to mask parts of a layer, Widget, or drawing:

1. **Radius of Rectangles**: Set the `radius` style property or the `radius` in the draw descriptors. This creates rounded rectangles, borders, outlines, etc.. However, the content of subsequent renderings will not be masked out in the corners.

2. **Clip Radius of Images**: Similar to rectangles, images can also be rendered with a `radius`. Since layer drawing and image drawing are handled the same way, this works for layers as well.

   You can draw various content on a layer and then render the layer with a `clip_radius`, masking out all the content on the corners.

3. **Rectangle Mask Draw Task**: A special Draw Task can mask out a rectangle from a layer by setting the alpha channel of certain pixels to 0. To achieve this:

   - Create an `lv_draw_mask_rect_dsc_t` descriptor.
   - Set `area`, `radius`, and `keep_outside` parameters. If `keep_outside` is set to 1, areas outside of `area` remain unchanged. Otherwise, they are cleared.
   - Call `lv_draw_mask_rect(layer, &dsc)`.

   Note: The layer must have a color format with an alpha channel, typically `LV_COLOR_FORMAT_ARGB8888`.

   In most cases, the *"Clip Radius of Images"* method is better because it blends the layer with a radius mask on the fly, avoiding a dedicated masking step. However, the *"Rectangle Mask Draw Task"* is useful when multiple areas need clearing or when the area to be masked differs from the layer area.

4. **Clip Corner Style Property**: Enabling `..._style_clip_corner` in a local or global style allows LVGL to create a layer for the top and bottom corner areas of a Widget. It renders the children there and blends it by setting `clip_radius` to the layer.

5. **Bitmap Masking for Images**: Using `..._style_bitmap_mask` or `bitmap_mask` in `lv_draw_image_dsc_t` allows setting an A8 or L8 image as a mask for an image/layer during blending.

   - Limitation: The mask always aligns to the center, and only one bitmap mask can be used for an image/layer.
   - When `..._style_bitmap_mask` is used, LVGL automatically creates a layer, renders the Widgets there, and applies the bitmap mask during blending.
   - Alternatively, the `bitmap_mask` property in the draw descriptor can be used directly for image drawing.

   By using the Canvas Widget with an `LV_COLOR_FORMAT_L8` buffer, bitmap masks can be rendered dynamically.

API

lv_draw.h

lv_draw_arc.h

lv_draw_image.h

lv_draw_label.h

lv_draw_line.h

lv_draw_mask.h

lv_draw_rect.h

lv_draw_triangle.h

lv_grad.h

CAPITOLO 8

XML - Declarative UI

8.1 Panoramica

8.1.1 Introduzione

LVGL is capable of loading UI elements written in XML. The XML file can be written by hand, but it's highly recommended to use LVGL's UI |nbsp| Editor to write the XML files. The UI Editor provides features like:

- Instant preview of the XML files
- Autocomplete and Syntax highlighting
- Online preview for collaboration and testing
- Figma integration to easily reimplement the designs

> **Avvertimento**
>
> The UI Editor and the XML loader are still under development and not production-ready. Consider them as an open beta, or experimental features.

Describing the UI in XML in a declarative manner offers several advantages:

- XML files can be loaded at runtime (e.g., from an SD card) to change the application build.
- XML is simpler to write than C, enabling people with different skill sets to create LVGL UIs.
- XML is textual data, making it easy to parse and manipulate with a large number of programming and scripting languages.
- XML can be used to generate LVGL code in any language.
- XML helps to separate the view from the internal logic.

Concept

The XML files are Component-oriented. To be more specific, they are `Component-Library` oriented. That is, they are structured in a way to make it easy to create reusable Component Libraries.

For example, a company can have a Component Library for the basic Widgets for all its products (generic widgets for all smart devices), and create other industry-specific Libraries (smart-home specific, smart-watch specific, etc.) containing only a few extra Widgets and Components.

These Component Libraries are independent, can be reused across many products, and can be freely versioned and managed.

Imagine a Component Library as a collection of XML files that describe Widgets, Components, Screens, images, fonts, and other assets stored in a git repository, which can be a submodule in many projects.

If someone finds a bug in the Component Library, they can just fix it in one place and push it back to the git repository so that other projects can be updated from it.

The built-in Widgets of LVGL are also considered a `Component Library` which is always available.

A UI Editor project can have any number of Component Libraries but will always have at least 2:

- LVGL's built-in Widgets, and
- XML-based definitions of Screen contents, along with other project-specific Components.

Widgets, Components, and Screens

It is important to distinguish between *Widgets*, *Components*, and *Screens*.

I Widget

Widgets are the core building blocks of the UI and are **not meant to be loaded at runtime** but rather compiled into the application. The main characteristics of Widgets are:

- In XML, they start with a `<widget>` root element.
- They are similar to LVGL's built-in Widgets.
- They are built using `lv_obj_class` objects.
- They have custom and complex logic inside.
- They cannot be loaded from XML at runtime because the custom code cannot be loaded.
- They can have a large API with `set/get/add` functions.
- They can themselves contain Widgets as children (e.g., `Tabview`'s tabs, `Dropdown`'s lists).

Any handwritten Widget can be accessed from XML by:

1. Defining its API in an XML file.
2. Writing and registering an XML parser for it. See some examples here.

Components

Components are built from other Components and Widgets, and **can be loaded at runtime**. The main characteristics of Components are:

- In XML, they start with a `<component>` root element.
- They are built in XML only and cannot have custom C code.
- They can be loaded from XML as they don't contain custom C code, only XML.
- They are built from Widgets or other Components.
- They can be used for styling Widgets and other Components.
- They can contain (as children) Widgets or other Components.
- They can have a simple API to pass properties to their children (e.g., `btn_text` to a Label's text).

Regardless of whether the XML was written manually or by the UI Editor, the XML files of Components can be registered in LVGL, and after that, instances can be created. In other words, LVGL can just read the XML files, "learn" the Components from them, and thereafter create children as part of Screens and other Components.

Screen

Screens are similar to Components:

- In XML, they start with a `<screen>` root element.
- They are built from Widgets or other Components to describe the *Screen*.
- They can be loaded from XML at runtime as they describe only visual aspects of the UI.
- They do not have an API.
- They can be referenced in Screen load events.

Syntax Teaser

Each Widget, Component, or Screen XML file describes a single UI element. The root element for Widgets, Components, and Screens is `<widget>`, `<component>`, and `<screen>`, respectively. Other than that, the contained XML elements are almost identical. This is a high-level overview of the most important XML elements that will be children of these root elements:

<api>
Describes the properties that can be `set` for a Widget or Component. Properties can be referenced by `$`. For Widgets, custom enums can also be defined with the `<enumdef>` tag.

<consts>
Specifies constants (local to the Widget or Component) for colors, sizes, and other values. Constant values can be referenced using `#`.

<styles>
Describes style (`lv_style_t`) objects that can be referenced (and shared) by Widgets and Components later.

<view>
Specifies the appearance of the Widget or Component by describing the children and their properties.

This is a simple example illustrating what an LVGL XML Component looks like. Note that only the basic features are shown here.

```
<component>
    <consts>
        <px name="size" value="100"/>
        <color name="orange" value="0xffa020"/>
    </consts>

    <api>
        <prop name="btn_text" default="Apply" type="string"/>
    </api>

    <styles>
        <style name="blue" bg_color="0x0000ff" radius="2"/>
        <style name="red" bg_color="0xff0000"/>
    </styles>

    <view extends="lv_button" width="#size">
        <style name="blue"/>
        <style name="red" selector="pressed"/>
        <my_h3 text="$btn_text"
```

(continues on next page)

(continua dalla pagina precedente)

```
                color="#orange"
                align="center"/>
    </view>
</component>
```

Usage Teaser

LVGL's UI Editor can be used in two different ways.

Export C and H Files

The Widgets, Components, Screens, images, fonts, etc., can be converted to .C/.H files having plain LVGL code. The exported code works the same way as if it was written by the user.

In this case, the XML files are not required anymore to run the C code (unless modifications may be made later and code is exported again).

The XML files were used only during editing/implementing the Widgets and Components to save recompilation time and optionally leverage other useful UI Editor features.

Load the UI from XML

Although the Widgets' code always needs to be exported in C and compiled into the application (just like the built-in LVGL Widgets are also part of the application), the Components' XML can be loaded and any number of instances can be created at runtime.

In the simplest case, a Component can be registered with `lv_xml_component_register_from_file(path)` and an instance can be created with `lv_obj_t * obj = lv_xml_create(parent, "my_button", NULL)`.

Note that loading the UI from XML practically has no impact on performance. Once the XML files are registered and the UI is created, it behaves the same way as if it were created from C code.

Registering XMLs and creating instances is not memory hungry nor slow. The biggest memory overhead is that the `<view>` of the Components is saved in RAM (typically 1–2 kB/component).

8.1.2 Component Library

Panoramica

The collection of Components, Widgets, Screens, Images, Fonts, etc., is called a Component Library.

A Component Library can be fully self-sufficient, but it can also reference data from other Component Libraries.

LVGL itself is a Component Library that supplies the built-in Widgets, data types, etc. You can find the XML files that describe the LVGL Widgets here.

A project always has at least 2 Component Libraries: that of LVGL as mentioned above, and its own where the Screens, Components, and Widgets of the project are defined. A project may include additional Component Libraries.

Structure

A typical structure for a Component Library looks like this:

```
name_of_the_component_library
├── globals.xml
├── components
│   ├── component1.xml
│   ├── component2.xml
│   └── other_folder
```

(continues on next page)

(continua dalla pagina precedente)

```
│       ├── component3.xml
│       └── component4.xml
├── widgets
│   ├── widget1
│   │   ├── widget1.xml
│   │   ├── widget1.c
│   │   ├── widget1.h
│   │   ├── widget1_gen.c
│   │   ├── widget1_gen.h
│   │   ├── widget1_private_gen.h
│   │   └── widget1_xml_parser.c
│   └── widget2
│       └── ...same as widget1...
├── screens
│   ├── screen1.xml
│   └── screen2.xml
├── fonts
│   ├── font1.ttf
│   └── font2.ttf
└── images
    ├── image1.png
    └── image2.png
```

Visibility

The content of all `globals.xml` files is part of a common global scope, and any Components, Widgets or Screens defined therein can be used in all .XML files.

Styles, constants, and other data defined in the XML file of Components, Widgets, or Screens are local to that XML file.

Thus, there are two namespaces:

1. **local namespace** within the given XML file of Components, Widgets and Screens.
2. **global namespace** created from the data in the `globals.xml` file from each Component Library included.

To find names referenced in XML files, the local namespace is checked first, and if a name is not found there, then the global namespace is checked.

The names of defined Components, Widgets and Screens become part of the global namespace and must be unique therein. This ensures that each Component has a unique name.

All data belonging to the LVGL core Component Library is prefixed by `lv_` (e.g., `lv_label`, `lv_font_default`).

A custom Component can be prefixed with `watch_`, `small_`, `light_`, or anything else the developer deems appropriate.

LVGL's UI Editor will show an error if there is a name conflict.

`globals.xml`

A single `globals.xml` file should be created in the root directory of each Component Library. The definitions in it do not belong to any specific Widget but are available throughout the entire UI, Widgets, and all XML files. The valid tags in it are:

<config>
: Can specify name and help.

<api>
Used with `<enumdefs>` to show possible values for Widget or Component attributes.

<subjects>
List of *Subjects*. Can be considered the API of a Component Library.

<consts>
Globally available constants.

<styles>
Globally available styles.

<fonts>
Globally available fonts.

<images>
Globally available images.

<const_variants>
Vedi di seguito.

<style_variants>
Vedi di seguito.

The `globals.xml` must be the only `globals.xml` file within the Component Library's folder tree.

From each `globals.xml` file, a `<config.name>.h` file is generated, which is included by all generated header files --- not only in the sub-folders where `globals.xml` is created, but in all exported .C and .H files. This ensures that constants, fonts, and other global data are available for all Widgets and new Widgets.

Esempio

A `globals.xml` file of a Component Library might look like this:

```
<globals>
    <api>
        <enumdef name="mode">
            <enum name="slow"/>
            <enum name="fast"/>
        </enumdef>
    </api>

    <consts>
        <px name="small_unit" value="8"/>
        <px name="large_unit" value="16"/>
    </consts>

    <styles>
        <style name="style_red" bg_color="0xff0000">
    </styles>

    <images>
        <file name="arrow_left" src="A:/images/arrow_left.png"/>
    </images>

    <fonts>
        <tiny_ttf name="big" src="A:/fonts/arial.ttf" size="28"/>
    </fonts>
</globals>
```

8.1.3 Project

Panoramica

A single `project.xml` file should be created for each project where the following content is specified:

<folders>
Specifies the path to Component Libraries. LVGL's base Widgets are always loaded automatically.

<targets>
Describes various hardware configurations, allowing the UI Editor to check if the UI is out of resources and to select different previews for each Screen according to the specified displays.

In LVGL's Editor the `folders`, `memory` and `` <display color_format="..."/>`` are not supported yet.

Esempio

```
<project>
    <folders>
        <folder path="../widget_lib1"/>
        <folder path="/home/user/work/ui_libs/modern"/>
        <folder path="https://github.com/user/repo"/>
    </folders>

    <targets>
        <target name="small">
            <display width="320" height="240" color_format="RGB565"/>
            <memory name="int_ram" size="128K"/>
            <memory name="ext_ram" size="2M"/>
            <memory name="int_flash" size="512K"/>
            <memory name="ext_flash" size="32M"/>
        </target>

        <target name="large">
            <display width="1024" height="768" color_format="XRGB8888"/>
            <memory name="int_ram" size="128K"/>
            <memory name="ext_ram" size="2M"/>
            <memory name="int_flash" size="512K"/>
            <memory name="ext_flash" size="32M"/>
        </target>
    </targets>
</project>
```

8.1.4 Syntax

Naming conventions

- A standard XML syntax is used.
- Lowercase letters with _ separation are used for attribute names.
- Tag names follow the usual variable-name rules: they must start with a letter or `'_'` and the rest of the name may be comprised of letters, `'_'` and digits.
- The LVGL API is followed as much as possible, e.g., `align="center"`, `bg_color="0xff0000"`.
- For colors, all these syntaxes are supported (similar to CSS colors): `0x112233`, `#112233`, `112233`, `0x123`, `#123`, `123`. Note that like CSS, `0x123`, `#123` and `123` all mean `#112233`.
- `params` can be referenced with `$`.
- `consts` can be referenced with `#`.

- `styles` can be attached to states and/or parts like this: `styles="style1 style2:pressed style3:focused:scrollbar"`.
- Local styles (i.e. styles that are stored within the Component and thus not shared by any other Components) can be used like this: `<lv_label style_text_color="0xff0000" style_text_color:checked="0x00ff00"/>`.

Types

All of the types can be used as API property types, but only a subset of them can be used as constant and *Subject* types.

Simple types

The following simple built-in types are supported:

bool
a `true` or `false`.

int
Integer number in the range of roughly -2B to 2B by default. (Same as `int32_t` in C.)

px
Simple pixel units. The unit `px` can be omitted.

%
Percentage. `%` must be appended to the value as the unit. (Means the same as *`lv_pct()`*.)

content
Means `LV_SIZE_CONTENT`.

string
Simple NUL-terminated string. When multiple strings are used in a property or string array, `'` should be used. E.g. `foo="'a' 'b'"`.

color
A color stored as 24-bit RGB (*`lv_color_t`*).

opa
Opacity value in the range of 0 to 255 or 0 to 100%. Like CSS, percentage values must be be followed by '%'.

lv_obj
Pointer to a Widget (`lv_obj_t*`).

screen
Pointer to a screen (also `lv_obj_t*`).

time_ms
Means some time in milliseconds unit

deg_0.1
Degrees with 0.1 resolution

scale_1/256
Scale/Zoom, where 256 means 100%, 128 means 50%, 512 means 200% etc.

Name-based types

In XML files, fonts, images, styles, etc., are not used by pointer but by string names. For example, a style is defined like `<style name="red" bg_color="0xff0000"/>`. Later, they can be referenced by their names.

This means that the actual values need to be bound to the names when the UI is loaded from XML, otherwise, LVGL wouldn't know what a name means.

Most of these connections are done automatically (e.g., for styles, fonts, images, animations, gradients, etc.), but others need to be connected manually (e.g., event callbacks where the callback itself is available only in the code).

For fonts and images, the connections are created automatically if the source is a file. If the font or image is compiled into the application (as a C array), the user needs to specify which variable a given name refers to.

To create these connections, functions like

- `lv_xml_register_image(scope, name, pointer)`
- `lv_xml_register_font(scope, name, pointer)`
- `lv_xml_register_event_cb(scope, name, callback)`
- ecc.

can be used. Later, a pointer to the object can be retrieved by

- `lv_xml_get_image(scope, name)`
- `lv_xml_get_font(scope, name)`
- `lv_xml_get_event_cb(scope, name)`
- ecc.

`scope` can be `NULL` to use the global scope or *`lv_xml_component_get_scope`*`("my_component")` returns the a component's local scope.

style
Name of a style. *`lv_xml_get_style_by_name`*`(&ctx, name)` returns an *`lv_style_t`*`*`.

font
Name of a font. *`lv_xml_get_font`*`(&ctx, name)` returns an `lv_font_t*`.

image
Name of an image. *`lv_xml_get_image`*`(&ctx, name)` returns an `const void*`, which can be *`lv_image_dsc_t`*`*` or a NUL-terminated string path to a file.

animation
Name of an animation descriptor. `lv_xml_get_anim(&ctx, name)` returns an `lv_anim_t*`.

subject
Name of a *Subject*. *`lv_xml_get_subject`*`(&ctx, name)` returns an *`lv_subject_t`*`*`.

grad
Name of a gradient. `lv_xml_get_grad(&ctx, name)` returns an *`lv_grad_dsc_t`*`*`.

event_cb
Name of an event callback. *`lv_xml_get_event_cb`*`(&ctx, name)` returns an *`lv_event_cb_t`*.

screen_create_cb
In XML it's the name of a screen XML file. In exported code it's a function like `lv_obj_t * my_screen_create(void)`

Arrays

An array of any type can be defined in four ways:

int[N]
An integer array with `N` elements. In the exported code `N` is passed a parameter after the array.

string[...NULL]
An array terminated with a `NULL` element. `NULL` can be replaced by any value.

string[5]
An array that must have exactly 5 elements. In the exported code only the array will be passed as the Widget assumes the given number of elements.

string[]
No `NULL` termination and no count parameter, used when the number of elements is not known or delivered via another mechanism, such as via a function parameter.

Enums

`<enumdef>` can be used in the `<api>` tags to create custom enums for **Widgets**. This is not supported for Components.

Ad esempio:

```
<api>
    <enumdef name="my_widget_mode" help="Possible modes" help-zh="Chinese␣
↪help">
        <enum name="normal" help="Normal mode" help-zh="Normal mode in␣
↪Chinese" value="0x10"/>
        <enum name="inverted" help="Inverted mode"/>
    </enumdef>

    <prop name="mode" help="help"type="enum:my_widget_mode" help="help"/>
</api>
```

When used as a type, a + suffix means multiple values can be selected and ORed. For example: `type="axis+"`. In this case, the options should be separated by `|`, for example: `axis=primary_x|secondary_y`.

Compound types

Types can be compound, meaning multiple options/types are possible. For example, for width: `type="px|%|content"`.

Limiting accepted values

It is also possible to limit the possible options the user can select from an enum. Ad esempio:

- Enums: `type="dir(top bottom)"`
- Colors: `type="color(0xff0000 0x00ff00 0x0000ff)"`
- Strings: `type="string('Ok' 'Cancel')`

Limiting accepted values is not supported yet, however in the UI Editor if an invalid option is selected, it will be highlighted as an error.

8.2 Build UIs

8.2.1 Root elements

Components

Panoramica

Components are one of the main building blocks for creating new UI elements.

`<component>`s support the following child XML tags:

- *<consts>*
- *<api>*

- *<styles>*, and
- *<view>*
- *<previews>*

Although they can't contain C code, they are very powerful:

- They can extend another Component or Widget (the base can be defined)
- Components can be built from Widgets and other Components
- A custom API can be defined
- Local styles can be defined, and the global styles can be used
- Local constants can be defined, and the global constants can be used
- Function calls, subject changes, or screen load/create events can be added. See *XML Events*
- Previews can be defined to preview the components in various settings

Unlike Widgets (which are always compiled into the application), Components can either:

1. be loaded at runtime from XML, or
2. be exported to C code and compiled with the application.

Usage from XML

In XML Files

Using Components in XMLs is very intuitive. The name of the components can be used as XML tag in the `<view>` of other Components, Screens, and Widgets.

The Component properties are just XML attributes.

To load Components from file, it's assumed that the XML files are saved to the device either as data (byte array) or as file. Once the data is saved, each component can be registered, and instances can be created after that.

```
<!-- my_button.xml -->
<component>
    <view extends="lv_button" flex_flow="row">
        <lv_image src="logo"/>
        <my_h3 text="Title"/>
    </view>
</component>
```

Styles, *Constants*, and *custom API* can also be described in the XML files.

Registration

Once a Component is created (e.g., `my_button`), it can be registered by calling either:

- *`lv_xml_component_register_from_file`*`("A:lvgl/examples/others/xml/my_button.xml")`
- *`lv_xml_component_register_from_data`*`("my_button", xml_data_of_my_button)`

These registration functions process the XML data and store relevant information internally. This is required to make LVGL recognize the Component by name.

Note that the "A:" in the above path is a file system "driver identifier letter" from *File System (lv_fs_drv)* and used accordingly. See that documentation for details.

When loaded from a file, the file name is used as the Component name.

During registration, the `<view>` of the Component is saved in RAM.

Instantiation

After registration, a new instance of any registered Component can be created with:

```
lv_obj_t * obj = lv_xml_create(lv_screen_active(), "my_button", NULL);
```

The created Widget is a normal LVGL Widget that can be used like any other manually-created Widget.

The last parameter can be `NULL` or an attribute list, like this:

```
/* Can be local */
char * my_button_attrs[] = {
    "x", "10",
    "y", "-10",
    "align", "bottom_left",
    NULL, NULL,
};

lv_obj_t * btn1 = lv_xml_create(lv_screen_active(), "my_button", my_button_
↪attrs);
```

Usage from Exported Code

From each Component XML file, a C and H file is exported with a single function inside:

```
lv_obj_t * component_name_create(lv_obj_t * parent, ...api properties...);
```

where 'component_name' (in the function above) is replaced by the Component's XML file name.

When a Component is used in another Component's XML code and the code is exported, this `create` function will be called. This means that Components do not have a detailed set/get API but can be created with a fixed set of parameters.

If the user needs to access or modify values dynamically, it is recommended to use a *Data bindings via Subject*.

The user can also call these `..._create()` functions at any time from application code to create new components on demand.

Extending

Additionally, when a Component is created, it can use the extended Widget's attributes (see `<view extends=". .">` in the code examples below).

This means that Components inherit the API of the extended Widget as well.

Custom Properties

The properties of child elements can be adjusted, such as:

```
<my_button x="10" width="200"/>
```

However, it's also possible to define custom properties in the `<api>` tag. The properties then can be passed to any properties of the children by referencing them by `$`. Ad esempio:

```
<!-- my_button.xml -->
<component>
    <api>
        <prop name="btn_text" type="string"/>
    </api>
```

(continues on next page)

(continua dalla pagina precedente)

```
    <view extends="lv_button">
        <lv_label text="$btn_text"/>
    </view>
</component>
```

And it can be used like

```
<!-- my_list.xml -->
<component>
    <view>
        <my_button btn_text="First"/>
        <my_button btn_text="Second"/>
        <my_button btn_text="Third"/>
    </view>
</component>
```

In this setup, the `btn_text` property is mandatory, however it can be made optional by setting a default value:

```
<prop name="btn_text" type="string" default="Title"/>
```

See *<api>* for more details and *XML Syntax* for all the supported types.

Esempi

The following example demonstrates parameter passing and the use of the `text` label property on a Component. Styles and Constants are also shown.

```
<!-- h3.xml -->
<component>
    <view extends="lv_label" style_text_color="0xffff00"/>
</component>
```

```
<!-- red_button.xml -->
<component>
    <api>
        <prop type="string" name="btn_text" default="None"/>
    </api>

    <consts>
        <int name="thin" value="2"/>
    </consts>

    <styles>
        <style name="pressed_style" border_width="#thin" border_color=
↪"0xff0000"/>
    </styles>

    <view extends="lv_button" style_radius="0" style_bg_color="0xff0000">
        <style name="pressed_style" selector="pressed"/>

        <h3 text="Some text"/>
        <h3 text="$btn_text" y="40"/>
    </view>
</component>
```

```
lv_xml_component_register_from_file("A:path/to/h3.xml");
lv_xml_component_register_from_file("A:path/to/red_button.xml");

/* Creates a button with "None" text */
lv_xml_create(lv_screen_active(), "red_button", NULL);

/* Use attributes to set the button text */
const char * attrs[] = {
    "btn_text", "Click here",
    NULL, NULL,
};
lv_xml_create(lv_screen_active(), "red_button", attrs);
```

Live Example

Load components at runtime

```
#include "../../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_XML

void lv_example_xml_1(void)
{
    /*A red button created from builti-in LVGL widgets
     *It has an API parameter too to change its text.*/
    const char * red_button_xml =
        "<component>"
        "  <api>"
        "    <prop name=\"button_text\" type=\"string\" default=\"None\"/>"
        "  </api>"
        "  <view extends=\"lv_button\" radius=\"0\" style_bg_color=\
↪"0xa91500\">"
        "    <lv_label text=\"$button_text\" align=\"center\"/>"
        "  </view>"
        "</component>";

    /*The card is just an lv_obj where a label and two red buttons are␣
↪used.
     * Its API allow setting a title (label test) and the action (the text␣
↪of a button)*/
    const char * card_xml =
        "<component>"
        "  <api>"
        "    <prop name=\"title\" type=\"string\" default=\"Hello world\"/>
↪"
        "    <prop name=\"action\" type=\"string\"/>"
        "  </api>"
        "  <view width=\"200\" height=\"content\">"
        "    <lv_label text=\"$title\" align=\"top_mid\"/>"
        "    <red_button y=\"20\" align=\"top_left\" button_text=\"Cancel\
↪"/>"
        "    <red_button y=\"20\" align=\"top_right\" button_text=\"
↪$action\"/>"
        "  </view>"
        "</component>";

    /* Motor card is a special case of a card where the title and action␣
```

(continues on next page)

(continua dalla pagina precedente)

```
↪are already set*/
    const char * motor_card_xml =
        "<component>"
        "  <view extends=\"card\" title=\"Motor start?\" action=\"Start\">"
        "  </view>"
        "</component>";

    /*Register all the custom components*/
    lv_xml_component_register_from_data("red_button", red_button_xml);
    lv_xml_component_register_from_data("card", card_xml);
    lv_xml_component_register_from_data("motor_card", motor_card_xml);

    lv_obj_t * card;
    /*Create a card with the default values*/
    card = (lv_obj_t *) lv_xml_create(lv_screen_active(), "card", NULL);

    /*Create a motor card too. The returned value can be adjusted freely*/
    card = (lv_obj_t *) lv_xml_create(lv_screen_active(), "motor_card",␣
↪NULL);
    lv_obj_set_y(card, 90);

    /*Pass properties to a card*/
    const char * attrs[] = {
        "y", "180",
        "action", "Apply",
        "title", "New title",
        NULL, NULL,
    };
    card = (lv_obj_t *) lv_xml_create(lv_screen_active(), "card", attrs);

}
#endif
```

```
#include "../../lv_examples.h"
#if LV_BUILD_EXAMPLES && LV_USE_XML && LV_USE_TRANSLATION

void lv_example_xml_2(void)
{
    lv_result_t res;
    res = lv_xml_component_register_from_file("A:lvgl/examples/others/xml/
↪my_h3.xml");
    if(res != LV_RESULT_OK) {
        lv_obj_t * label = lv_label_create(lv_screen_active());
        lv_label_set_text(label, "Couldn't open the XML files.");
        lv_obj_center(label);
        return;
    }
    lv_xml_component_register_from_file("A:lvgl/examples/others/xml/my_
↪card.xml");
    lv_xml_component_register_from_file("A:lvgl/examples/others/xml/my_
↪button.xml");
    lv_xml_component_register_from_file("A:lvgl/examples/others/xml/view.
↪xml");
    lv_xml_translation_register_from_file("A:lvgl/examples/others/xml/
↪translations.xml");
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_xml_register_font(NULL, "lv_montserrat_18", &lv_font_montserrat_18);

    lv_translation_set_language("de");

    lv_obj_t * obj = (lv_obj_t *) lv_xml_create(lv_screen_active(), "view",
↪ NULL);
    lv_obj_set_pos(obj, 10, 10);

    lv_xml_component_unregister("my_button");

    const char * slider_attrs[] = {
        "x", "200",
        "y", "-15",
        "align", "bottom_left",
        "value", "30",
        NULL, NULL,
    };

    lv_obj_t * slider = (lv_obj_t *) lv_xml_create(lv_screen_active(), "lv_
↪slider", slider_attrs);
    lv_obj_set_width(slider, 100);
}
#endif
```

API

I Widget

Panoramica

Besides components and Screens, Widgets are the other main building blocks of UIs. The XML file of the Widgets is wrapped in a `<widget>` XML root element.

`<widget>`s support the following child XML tags:

- *<consts>*
- *<api>*
- *<styles>*, and
- *<view>*
- *<previews>*

Just like Components, Widgets also can be the children of other Widgets and Components.

The main difference is that Widgets are written in C and compiled into the application. It means unlike Components, Widgets can contain custom C code. For example, when a property is set, any complex C code can run to set that value.

To connect the C code to XML, XML parser functions need to be implemented and registered. It's pretty simple as LVGL already provides all the helper functions and the required libraries. Also there are many XML parser examples here.

Built-in Widgets

The built-in LVGL widgets (e.g. *Slider (lv_slider)*, *Label (lv_label)*, *Chart (lv_chart)*, etc.) already have XML parsers and therefore are available in XML.

Ad esempio:

```
<view>
    <lv_label x="10" text="Hello"/>
</view>
```

The built-in widgets are

- pure C code (e.g. lv_slider.c)
- an XML file to define only the API (e.g. lv_slider.xml). It is used only in the UI Editor to validate and autocomplete properties.
- an XML parser C file to map the XML attributes to C functions. (e.g. lv_xml_slider_parser.c)

Creating New Widgets

It's possible to create new widgets in the same way as the built-in LVGL widgets are handled.

However, using the UI Editor it's much faster and simpler. When an XML file is created and the `<widget>` root element is used, the following .C/.H files are generated automatically:

<widget_name>_gen.h
Contains the generated API implementation of the widget (overwritten on each code export).

<widget_name>_private_gen.h
Contains private API and the data for the widget (overwritten on each code export).

<widget_name>_gen.c
Contains the internals of the Widget, e.g. constructor with the children, destructors, event handler, etc. (overwritten on each code export).

<widget_name>.h
Includes `<widget_name>_gen.h` and allows the user to define custom APIs. Only a skeleton is exported once.

<widget_name>.c
Contains hooks from `<widget_name>_gen.c` and allows the user to write custom code. Only a skeleton is exported once.

<widget_name>_xml_parser.c
Processes the XML strings and calls the required functions according to the set attributes. Only a skeleton is exported once.

Utilizzo

XML Parser

To make the Widgets accessible from XML, an XML parser needs to be created and registered for each Widget. The XML parser for the label Widget looks like this:

```
void * lv_xml_label_create(lv_xml_parser_state_t * state, const char **␣
↪attrs)
{
    /* Create the label */
    void * obj = lv_label_create(lv_xml_state_get_parent(state));
    return obj;
}

void lv_xml_label_apply(lv_xml_parser_state_t * state, const char ** attrs)
{
    void * obj = lv_xml_state_get_item(state);

    /* Apply the common properties, e.g., width, height, styles, flags,␣
```

(continues on next page)

(continua dalla pagina precedente)

```
↪etc. */
    lv_xml_obj_apply(state, attrs);

    /* Process the label-specific attributes */
    for(int i = 0; attrs[i]; i += 2) {
        const char * name = attrs[i];
        const char * value = attrs[i + 1];

        if(lv_streq("text", name)) lv_label_set_text(obj, value);
        if(lv_streq("long_mode", name)) lv_label_set_long_mode(obj, long_
↪mode_text_to_enum(value));
        /* Process more props here */
    }
}

/* Helper to convert strings to enum values */
static lv_label_long_mode_t long_mode_text_to_enum(const char * txt)
{
    if(lv_streq("wrap", txt)) return LV_LABEL_LONG_WRAP;
    if(lv_streq("scroll", txt)) return LV_LABEL_LONG_SCROLL;

    LV_LOG_WARN("%s is an unknown value for label's long_mode", txt);
    return 0; /* Return 0 in the absence of a better option. */
}
```

A Widget XML processor can be registered like this:

```
lv_xml_widget_register("lv_label", lv_xml_label_create, lv_xml_label_
↪apply);
```

After registration, a Widget can be created like this from C code:

```
const char * attrs[] = {
    "text", "Click here",
    "align", "center",
    NULL, NULL,
};

lv_xml_create(lv_screen_active(), "lv_label", attrs);
```

And in XML it can be used like

```
<view>
    <lv_label width="100" text="I'm a label!" wrap="scroll"/>
</view>
```

Adding Custom Code

`<widget_name>.c` contains three hooks:

- **Constructor hook**: Called when the Widget and all its children are created. Any modifications can be done on the children here.
- **Destructor hook**: Called when the Widget is deleted. All manually allocated memory needs to be freed here.
- **Event hook**: Called at the beginning of the Widget's event callback to perform any custom action.

In this C file, the `set` functions for each API `<prop>` also need to be implemented. The declaration of these functions is automatically exported in `<widget_name>_gen.h`.

Besides these, any custom code and functions can be freely implemented in this file.

Elements

Elements are internal parts of the widget that can be accessed and/or created dynamically. For example, tabs of a tabview, list of a dropdown, series of a chart, etc.

Just like any other Widget API properties, Elements also can be defined in the `<api>` tag of the Widget's XML.

Learn more about the Elements in the documentation page of *<api>*.

Screen

Panoramica

Screens work very similarly to Components. Both can be:

- Loaded from XML
- Contain Widgets and Components as children

Screens are wrapped in the `<screen>` XML root element and used to organize the content of the UI.

Screens can have only the following child XML tags:

- *<consts>*
- *<styles>*, and
- *<view>*

That is, Screens **cannot** have an *<api>* or *<preview>*.

Utilizzo

Each XML file describes a *Screen*. The name of the XML file will also be the name of the Screen.

This example illustrates a screen in XML. In the example, a `<my_header>` and a `<my_main_cont>` component is used to keep the screen simple.

```
<screen>
    <consts>
        <string name="title" value="Main menu"/>
    </consts>

    <styles>
        <style name="dark" bg_color="0x333"/>
    </styles>

    <view>
        <my_header label="#title"/>
        <my_main_cont>
            <style name="dark"/>

            <button text="Weather" icon="cloudy"/>
            <button text="Messages" icon="envelope"/>
            <button text="Settings" icon="cogwheel"/>
            <button text="About" icon="questionmark"/>
        </my_main_cont>
    </view>
</screen>
```

Code export

When the C code is exported from the UI Editor, `screen_name_gen.c/h` files are exported, containing only a single `lv_obj_t * screen_name_create(void)` create function.

By using this function, any number of screen instances can be created and loaded as needed.

Preview

Screens don't support the *<preview>* tag because it doesn't make sense to preview each screen in different resolutions.

As Screens are related to the target hardware in the `project.xml` file, multiple `<display>` elements can be defined. In the UI Editor, when a Screen is being developed, the user can select from all the defined displays in the Preview, and the Screen will be shown with the given resolution and color depth.

This is useful for verifying responsive designs.

Eventi

It's very common to load or create Screens on a button click or other events.

Both are supported by adding special XML tags as children of Components or Widgets:

```
<view>
    <lv_button>
        <lv_label text="Click or Long press me"/>

        <!-- Load an already created screen that has the name "first".
             Note that here the name of the instance is used,
             and not the name of the XML file. -->
        <screen_load_event screen="first" trigger="clicked" anim_type="fade
↪"/>

        <!-- Create an instance of "about" and load it.
             Note that here the name of the XML file is used. -->
        <screen_create_event screen="about" trigger="long_pressed"/>
    </lv_button>
</view>
```

Learn more on *XML Events* page.

8.2.2 Main tags

API

The `<api>` tag can be a child of `<widget>` and `<component>` tags. (`<screen>`s don't support custom APIs.)

The only common point is that both Widgets and Components support having `<prop>` (properties) in the `<api>` tag to describe their interface.

However, as Widgets and Components work very differently (Widgets have C code, but Components are pure XML), even properties are interpreted differently.

Components

Panoramica

While Widgets can have complex `set/get` APIs, Components are very simple.

When their XML is converted to a C file, only a `create` function is generated, where all the `<prop>`s are arguments. Ad esempio:

```
<api>
    <prop name="prop1" type="int"/>
    <prop name="prop2" type="string"/>
</api>
```

This generates the following C function:

```
lv_obj_t * my_component_create(lv_obj_t * parent, int32_t prop1, const␣
↪char * prop2);
```

These properties are set once (at creation time), and there are no specific `set` functions to modify the property later. LVGL's general API can still be used to modify any widget in the component, but no dedicated API functions are generated.

Referencing properties

`<prop>`s are simply forwarded to widget or component APIs. For example, if a component has `<prop name="button_label" type="string"/>`, it can be used in a label as `<lv_label text="$button_label"/>`.

In the generated code, these are passed as arguments in create/set functions.

Default values

Since each property is passed as an argument to the create function, each must have a value. This can be ensured by:

- Simply setting them in the XML instance
- Providing a default value in the `<api>`, e.g., `<prop name="foo" type="string" default="bar"/>`

Limitazioni

Note that none of the Widget API features such as `<param>`, `<enumdef>`, or `<element>` can be used for Components. Only simple properties that are forwarded are supported.

Esempio

```
<!-- my_button.xml -->
<component>
    <api>
        <prop name="button_icon" type="image" default="NULL"/>
        <prop name="button_label" type="string" default="Label"/>
    </api>

    <view extends="lv_button" flex_flow="row" width="100%">
        <lv_image src="$button_icon" inner_align="stretch" width="16"␣
↪height="16"/>
        <lv_label text="$button_label"/>
    </view>
</component>

<!-- my_list.xml -->
<component>
    <view flex_flow="column">
        <my_button button_label="First"/>
        <my_button button_label="Wifi" button_icon="img_wifi"/>
        <my_button button_label="Third"/>
```

(continues on next page)

(continua dalla pagina precedente)

```
    </view>
</component>
```

I Widget

Properties

Properties are the core part of describing a Widget's API.

```
<api>
    <prop name="text" type="string" help="Text of the label."/>
</api>
```

Parameters

Some properties take multiple parameters. For example: `lv_label_set_bind_text(label, subject, "%d °C")`

It's described as:

```
<api>
    <prop name="bind_text" help="Bind a subject's value to a label.">
        <param name="bind_text" type="subject" help="Integer or string␣
↪subject"/>
        <param name="fmt" type="string" help="Format string, e.g. %d °C "/>
    </prop>
</api>
```

And used as:

```
<lv_label bind_text="subject" bind_text-fmt="%d °C"/>
```

Parameters with the same name as the property can be referenced directly. Other parameters use `property-param` notation.

Unset parameters fall back to:

- Their default value (if defined)
- Type-specific defaults (e.g., 0, false, NULL)

Mapping

Each `<prop>` is mapped to a `set` function. This mapping is implemented in the Widget's XML parser. See the LVGL XML parsers.

If `<param>``s are used, they are passed to the same ``set` function. If a property is not set on a Widget instance, it is skipped and the Widget's built-in default is used.

<enumdef>

Only used with Widgets, this tag defines enums for parameter values.

```
<api>
    <enumdef name="my_widget_mode" help="Possible modes">
        <enum name="normal" help="Normal mode" value="0x10"/>
        <enum name="inverted" help="Inverted mode"/>
    </enumdef>
```

(continues on next page)

(continua dalla pagina precedente)

```
    <prop name="mode" help="Set Widget mode">
        <param name="mode" type="enum:my_widget_mode"/>
    </prop>
</api>
```

Enum values are ignored in export; the names are used and resolved by the compiler. XML parsers must handle mapping enum names to C enums.

<element>

Also exclusive to Widgets, elements define sub-widgets or internal structures (e.g., chart series, dropdown list, tab views).

They support `<arg>` and `<prop>`:

- `<arg>`s are required and used when creating/getting the element.
- `<prop>`s are optional and mapped to setters.

Elements are referenced as `<widget-element>` in views.

Name parts are separated by `-` (as `-` is not allowed inside names).

Element `access` types:

- `add`: Create multiple elements dynamically.
- `get`: Access implicitly created elements.
- `set`: Access indexed parts (e.g., table cells).
- `custom`: Map custom C function to XML.

As `add` and `get` elements return an object they also have a type. This type can be any custom type, for example, `type="my_data"`. In the exported code the return value will be saved in a `my_data_t *` variable.

If the type is `type="lv_obj"` it allows the element to have children widgets or components.

Note that, only the API can be defined in XML for elements; implementations must be in C.

access="add"

Elements are created via an `add` function:

```
<api>
    <element name="indicator" type="obj" help="The indicator of my_widget"␣
↪access="add">
        <arg name="color" type="color"/>
        <arg name="max_value" type="int"/>
        <prop name="value">
            <param name="value" type="int"/>
        </prop>
    </element>
</api>
```

Used in a view:

```
<my_widget width="100px">
    <my_widget-indicator name="indic1" color="0xff0000" max_value="120"␣
↪value="30"/>
</my_widget>
```

Generates:

```
lv_obj_t * my_widget_add_indicator(lv_obj_t * parent, lv_color_t color,␣
↪int32_t max_value);
void my_widget_set_indicator_value(lv_obj_t * obj, int32_t value);
```

access="get"

Used for internal/implicit elements:

```
<api>
    <element name="control_button" type="obj" help="A control button of my_
↪widget" access="get">
        <arg name="index" type="int"/>
        <prop name="title" type="string"/>
    </element>
</api>
```

Used in a view:

```
<my_widget width="100px">
    <my_widget-control_button name="btn1" index="3" title="Hello"/>
</my_widget>
```

Generates:

```
lv_obj_t * my_widget_get_control_button(lv_obj_t * parent, int32_t index);
void my_widget_set_control_button_title(lv_obj_t * obj, const char * text);
```

access="set"

Used for indexed access, like setting values in a table:

```
<api>
    <element name="item" type="obj" access="set">
        <arg name="index" type="int"/>
        <prop name="icon" type="img_src"/>
        <prop name="color" type="color"/>
    </element>
</api>
```

Used in a view:

```
<my_widget width="100px">
    <my_widget-item index="3" icon_src="image1" color="0xff0000"/>
</my_widget>
```

Generates:

```
void my_widget_set_item_icon(lv_obj_t * parent, int32_t index, const void␣
↪* icon_src);
void my_widget_set_item_color(lv_obj_t * parent, int32_t index, lv_color_t␣
↪color);
```

access="custom"

Used to describe any custom API functions with a custom name. "custom" elements can have only arguments and no `type` so they are pure setters.

```
<element name="bind_color" access="custom">
    <arg name="subject" type="subject"/>
    <arg name="new_color" type="color"/>
    <arg name="ref_value" type="int"/>
</element>
```

Used in a view:

```
<my_widget width="100px">
    <my_widget-bind_color subject="subject_1" color="0xff0000" ref_value=
↪"15"/>
</my_widget>
```

Generates:

```
void my_widget_bind_color(lv_obj_t * parent, lv_subject_t * subject, lv_
↪color_t color, int32_t ref_value);
```

Constants

Panoramica

Constants can be defined to replace any value with a selected type or to be used as special values.

The supported types are: - `color` - `px` - `percentage` - `string` - `opa` - `bool`

Utilizzo

```
<consts>
    <color name="color1" value="0xff0000" help="Primary color"/>
    <px name="pad_xs" value="8" help="Small padding"/>
</consts>
```

Constants can be used in:

- Style properties
- Widget and Component properties

And they can be used like this:

```
<styles>
    <style name="style1" bg_color="#color1"/>
</styles>
```

Stili

Panoramica

In XML files, both style sheets (*`lv_style_t`*) and local styles can be used.

Style Sheets

In the `<styles>` section, styles and their properties can be defined like this:

```
<style name="red"
    help="What is this style about?"
    border_width="2px"
    border_color="0xff0000"/>
```

Styles can be referenced like this in the `<view>`:

```
<view>
    <lv_slider value="20">
         <style name="main"/>
         <style name="red" selector="knob"/>
         <style name="blue" selector="knob focused"/>
    </lv_slider>
</view>
```

As shown in the example, parts and states can be set as `selector`.

Local Styles

Local styles can be used directly in a Widget, for example:

```
<lv_label style_bg_opa="200" style_bg_opa:disabled="100"/>
```

Gradients

Before the `<styles>` tag, the `<gradients>` tag can be used to describe various gradients, which can later be referenced in styles.

When a gradient is created, it can be referenced by its name, like:

```
<style bg_grad="grad1"/>
```

or

```
<lv_button style_bg_grad="grad1"/>
```

Horizontal or Vertical Gradient

To define a simple `<horizontal>` or `<vertical>` gradients:

```
<gradients>
    <horizontal name="grad1">
        <stop color="#ff0000" offset="20%" opa="40%"/>
        <stop color="#00ff00" offset="128" opa="100%"/>
    </horizontal>
</gradients>
```

Linear Gradient

To define a skewed gradient from two points:

```
<gradients>
    <linear name="grad1" start="50 50" end="100 80">
        <stop color="#ff0000" offset="20%" opa="100%"/>
        <stop color="#00ff00" offset="240" opa="100%"/>
    </linear>
</gradients>
```

Gradiente Radiale

To define a radial gradient:

```
<gradients>
    <radial name="grad1" center="100 50%" edge="200 50" focal_center="50 80
↪%" focal_edge="55 80%">
        <stop color="#ff0000" opa="100%"/>
        <stop color="#00ff00" opa="100%"/>
    </radial>
</gradients>
```

Gradiente Conico

To define a conical gradient:

```
<gradients>
    <conical name="grad1" center="80 50%" angle="45 270">
        <stop color="#ff0000" opa="100%"/>
        <stop color="#00ff00" opa="100%"/>
    </conical>
</gradients>
```

View

Panoramica

The `<view>` tag can be used in:

- `<component>`s
- `<widget>`s
- `<screen>`s
- `<test>`s

to describe how these items look. Inside `<view>`, children can be added in a nested way using already defined `widget`s and `component`s. Ad esempio:

```
<view>
    <lv_button width="200">
        <my_icon src="image1"/>
        <lv_label text="Click me"/>
    </lv_button>
</view>
```

Extends

The `<view>` itself is also a widget or component, which will become the parent of the children. To define the type of the `<view>`, use the `extends` attribute. For example, `extends="lv_slider"`. In this case, an `lv_slider` will be created first, and the children will be added to it.

By adding properties to the `<view>`, the extended type can be customized. Ad esempio:

```
<view extends="lv_slider" width="100%" style_bg_color="0xff8800" flex_flow=
↪"row">
    <style name="my_style" selector="pressed|knob"/>
    <lv_label text="Current value: "/>
    <lv_label bind_text="subject_1"/>
</view>
```

Rules for the allowed values of `extends`:

- `<component>`: can extend both `<widget>`s and `<component>`s

- `<widget>`: can extend only `<widget>`s
- `<screen>`: cannot extend anything
- `<test>`: can extend `<widget>`, `<component>`, or `<screen>`

Animazioni

TODO

Preview

Panoramica

In `<component>` and `<widget>`, it is possible to define `<preview>` tags. These are **not** exported to code and are **not** loaded from XML.

They are used only by the UI Editor to describe the context of the Component. For example, you might want to:

- Change the background of the Editor's preview to dark.
- Center the Component.
- Set margins.
- Change the size of the preview.

`style_radius` can be used to make preview rounded.

Utilizzo

You can think of a `<preview>` tag as an `lv_obj` where the following properties can be used:

- `width`, `height`
- Any local style properties, for example, `style_bg_color="0x333"`
- `flex` and `flex_flow`

It is also possible to define multiple previews, and in the UI Editor, you can select one of them.

Esempio

```
<component>
    <previews>

        <preview name="small_dark_round"
                 width="240" height="240"
                 style_bg_color="0x333" style_pad_all="32"
                 style_radius="32"/>

        <preview name="large_light"
                 width="1980" height="1080"
                 style_bg_color="0xeeeeee"/>

    </previews>

    <view>
        ...
    </view>

</component>
```

8.2.3 Assets

Immagini

Panoramica

An `<images>` section can be added to `globals.xml` files. If present, it describes how to map images with names.

Utilizzo

```
<images>
    <file name="avatar" src_path="avatar1.png"/>
    <data name="logo" src_path="logo1.png" color-format="rgb565" />
</images>
```

- `<file>` means that the image source is used as a file path:
- `<data>` means that the image is converted to a C array on export.

In both cases in the exported C code global `const void * <image_name>` variables are created and in the initialization function of the Component Library (e.g. `my_lib_init_gen()`) either the path or the pointer to the converted *`lv_image_dsc_t`* pointers are assigned to that variable.

In *`lv_image_set_src`*`(image, image_name)` `image_name` is used instead of the path or *`lv_image_dsc_t`* pointer.

Registering images

If the UI is created from XML at runtime and a `globals.xml` is parsed, the `<data>` tags are skipped because it is assumed that the user manually created the mapping. This is because the XML parser cannot automatically map an image like:

```
lv_image_dsc_t my_logo;
```

to

```
<data name="my_logo"/>
```

To register an image path or data in the XML engine use:

```
lv_xml_register_image(scope, "image_name", data)
```

`scope` is usually `NULL` to register the image in the global scope. To register an image locally for a component you can get its scope with:

```
lv_xml_component_get_scope("component_name")
```

After calling this function, when `"image_name"` is used as an image source in XML, `data` (can be a path or a pointer to an image descriptor) will be used.

Notes for the UI Editor

For simplicity, in the UI Editor's preview, images are always loaded as files. It makes the preview dynamic so no code export and compilation is needed when an image changes.

Fonts

Panoramica

A `<fonts>` section can be added in `globals.xml` files. Later, it might be supported in Components and Widgets to define local fonts and keep the global space cleaner.

Utilizzo

The following section creates a mapping between font names and their paths with various attributes:

```
<fonts>
    <bin as_file="false" name="medium" src="path/to/file.ttf" range="0x20-
↪0x7f" symbols="°" size="24"/>
    <tiny_ttf as_file="true" name="big" src_path="path/to/file.ttf" range=
↪"0x20-0x7f" symbols="auto" size="48"/>
    <freetype name="chinese" src_path="file.ttf" size="48" custom_freetype_
↪attribute="abc"/>
</fonts>
```

In `<styles>` and `<view>`, fonts can then be referenced by their name, e.g.,

```
<style name="style1" text_font="medium"/>
```

The tag name determines how the font is loaded. Currently, only `tiny_ttf as_file="true"` is supported.

- `bin`:
  - If `as_file="true"`: Converts the font file to `bin` (see `lv_font_conv`) which will be loaded by `lv_binfont_create()`.
  - If `as_file="false"` (default): On export, the font file will be converted to a C array LVGL font that can be used directly by LVGL.
- `tiny_ttf`:
  - If `as_file="true"`: Can be loaded directly by `lv_tiny_ttf_create_file()`.
  - If `as_file="false"` (default): The font file will be converted to a raw C array on export that will be loaded by `lv_tiny_ttf_create_data()`.
- `freetype`: The file can be loaded directly by `lv_freetype_font_create()`.

For simplicity, if `as_file="false"`, fonts will be loaded as files in the preview. Setting `as_file="false"` affects only the C export.

If the UI is created from XML at runtime and a `globals.xml` is parsed, the `as_file="false"` tags are skipped because it is assumed that the user manually creates the mapping. This is because the XML parser cannot automatically map an LVGL font definition like:

```
lv_font_t my_font_24;
```

to

```
<bin name="my_font_24"/>
```

Exported Code

When C code is exported, global `const lv_font_t * <font_name>` variables are created, and in the initialization function of the Component Library (e.g., `my_lib_init_gen()`), the actual font is assigned.

In `lv_style_set_text_font(&style1, <font_name>)`, the created font is referenced.

Default Font

`"lv_font_default"` can be used to access `LV_FONT_DEFAULT`. Other built-in fonts are not exposed by default.

Registering fonts

If the UI is created from XML at runtime and a `globals.xml` is parsed, the `<... as_file="false">` tags are skipped because it is assumed that the user manually created the mapping. This is because the XML parser cannot automatically map fonts like:

```
lv_font_t my_font;
```

to

```
<data name="my_font"/>
```

To register a font in the XML engine use:

```
lv_xml_register_font(scope, "font_name", &my_font);
```

`scope` is usually `NULL` to register the font in the global scope. To register a font locally for a component you can get its scope with:

```
lv_xml_component_get_scope("component_name");
```

After calling this function, when `"font_name"` is used as a font in XML, `&my_font` will be used.

Notes for the UI Editor

For simplicity, in the UI Editor's preview, fonts are always loaded as tinyttf fonts. It makes the preview dynamic as no code export and compilation is needed when a font changes.

8.2.4 Eventi

Panoramica

There are several ways to define events for user interactions. These events can be added as children of any widget.

Triggers

In all event types, the `trigger` attribute defines what kind of user action should trigger the event. All LVGL event types are supported with straightforward mapping:

- *LV_EVENT_ALL*: `"all"`
- *LV_EVENT_CLICKED*: `"clicked"`
- *LV_EVENT_PRESSED*: `"pressed"`
- ecc.

Call function

User-defined functions can be called like this:

```
<view>
    <lv_button width="200" height="100">
        <event_cb callback="my_callback_1" trigger="clicked" user_data=
↪"some_text"/>
        <lv_label text="Hello"/>
    </lv_button>
</view>
```

When the XML is loaded at runtime, the callback name needs to be mapped to a function using *lv_xml_register_event_cb*`("my_callback_1", an_event_handler)`.

The callback should follow the standard LVGL event callback signature: `void an_event_handler(lv_event_t * e);`

In the exported C code, it is assumed that a function with the exact name exists. For example, `callback="my_callback_1"` will be exported as:

```
void my_callback_1(lv_event_t * e); /* At the beginning of the exported␣
↪file */

lv_obj_add_event_cb(obj, my_callback_1, LV_EVENT_CLICKED, "some_text");
```

The `user_data` is optional. If omitted, `NULL` will be passed.

Screen Load and Create events

By using the `<screen_load_event>` and `<screen_create_event>` tags as a child of a widget or component, screens can be loaded or created on a trigger (e.g. click).

The difference between load and create is that:

- **load**: Just loads an already existing screen. After leaving the screen, it remains in memory, so all states are preserved.
- **create**: The screen is created dynamically, and when leaving the screen, it is deleted, so all changes are lost (unless they are saved in `subjects`).

Both tags support the following optional attributes:

- `trigger`: Event code that triggers the action (e.g. `"clicked"`, `"long_pressed"`, etc). Default: `"clicked"`.
- `anim_type`: Describes how the screen is loaded (e.g. `"move_right"`, `"fade_in"`). Default: `"none"`.
- `duration`: Length of the animation in milliseconds. Default: `0`. Only used if `anim_type` is not `"none"`.
- `delay`: Wait time before loading the screen in milliseconds. Default: `0`.

This is a simple example of both load and create:

```
<!-- screen1.xml -->
<screen>
   <view style_bg_color="0xff7788">
       <lv_button>
           <lv_label text="Create"/>
           <!-- Create an instance of "screen2" and load it. -->
           <screen_create_event screen="screen2" anim_type="over_right"␣
↪duration="500" delay="1000"/>
       </lv_button>
   </view>
</screen>

<!-- screen2.xml -->
<screen>
   <view style_bg_color="0x77ff88">
       <lv_button>
           <lv_label text="Load"/>
           <!-- Load an already created instance of screen1.-->
           <screen_load_event screen="screen1"/>
       </lv_button>
   </view>
</screen>
```

```
lv_xml_component_register_from_data("screen1", screen1_xml);
lv_xml_component_register_from_data("screen2", screen2_xml);

/*Create an instance of screen_1 so that it can loaded from screen2.*/
lv_obj_t * screen1 = lv_xml_create(NULL, "screen1", NULL);
lv_screen_load(screen1);
```

Set subject value

It's possible to set a *Subject* value on user interaction by adding a special child to any widget:

```
<view>
    <lv_button width="200" height="100">
        <subject_set_int_event    trigger="clicked" subject="subject_int" ␣
↪  value="10"/>
        <subject_set_float_event  trigger="clicked" subject="subject_float
↪"  value="12.34"/>
        <subject_set_string_event trigger="clicked" subject="subject_string
↪" value="Hello"/>
        <lv_label text="Set the values"/>
    </lv_button>
</view>
```

The usage is straightforward: the specified `subject` will be set to the given `value` when the `trigger` occurs.

Increment subject value

Incrementing or decrementing a *Subject* value can be defined as follows:

```
<view>
    <lv_button width="200" height="100">
        <subject_increment_event trigger="clicked" subject="subject_int1"␣
↪step="10"/>
        <subject_increment_event trigger="clicked" subject="subject_int2"␣
↪step="-10" min="0" max="50"/>
        <subject_increment_event trigger="clicked" subject="subject_float1
↪" step="2"/>
    </lv_button>
</view>
```

The `<subject_increment_event>` element defines a `step` to be added to the subject's current value when the `trigger` occurs. Optionally, `min` and/or `max` can be set to limit the subject's value.

`subject` must be an `int` or `float` subject.

If `step` is **negative**, the subject's value will be decremented. Only integer `step` values are supported now.

Note: Only integer subjects are supported by `<subject_increment>`.

8.2.5 Data binding (Subjects)

Panoramica

With the help of *Subjects*, the interface of the UI can be created.

A subject is a global data item whose value can be set either from the application or the UI, and whose value can be bound to widget properties.

For example, a `room1_temperature` subject's value can be set in the application when the temperature is measured, and can be bound to a label like this:

```
<lv_label bind_text="room1_temperature"/>
```

Defining subjects

Subjects can be created in `globals.xml` like this:

```
<globals>
    <subjects>
        <int name="battery_power" value="32"/>
        <string name="user_name" value="John"/>
    </subjects>
</globals>
```

As the example shows, a subject consists of a type, name, and initial value. Currently, only integer and string types are supported.

Simple binding

Some widgets (e.g., label, slider) support binding the subject's value directly to the widget. These bindings use attributes that start with `bind_*` and reference a subject.

```
<lv_slider bind_value="some_subject"/>
<lv_label bind_text="some_subject"/>
```

Once a binding is created, if the subject's value changes (e.g., by adjusting the slider), all bound widgets will be updated automatically.

Complex binding

In more complex cases—when a binding requires multiple parameters—the binding can be added as a child element of a widget. This allows binding multiple subjects with different parameters. Ad esempio:

```
<lv_label text="Hello world">
    <lv_obj-bind_flag_if_eq subject="subject1" flag="hidden" ref_value="10
↪"/>
    <lv_obj-bind_flag_if_gt subject="subject1" flag="clickable" ref_value=
↪"20"/>
</lv_label>
```

Explanation of complex bindings:

- `<lv_obj-bind_flag_if_eq>` — Set a flag if the subject's value **equals** the reference value.
- `<lv_obj-bind_flag_if_not_eq>` — Set a flag if the subject's value **does not equal** the reference value.
- `<lv_obj-bind_flag_if_gt>` — Set a flag if the subject's value is **greater than** the reference value.
- `<lv_obj-bind_flag_if_ge>` — Set a flag if the subject's value is **greater than or equal to** the reference value.
- `<lv_obj-bind_flag_if_lt>` — Set a flag if the subject's value is **less than** the reference value.
- `<lv_obj-bind_flag_if_le>` — Set a flag if the subject's value is **less than or equal to** the reference value.
- `<lv_obj-bind_state_if_eq>` — Set a state if the subject's value **equals** the reference value.
- `<lv_obj-bind_state_if_not_eq>` — Set a state if the subject's value **does not equal** the reference value.
- `<lv_obj-bind_state_if_gt>` — Set a state if the subject's value is **greater than** the reference value.

- `<lv_obj-bind_state_if_ge>` — Set a state if the subject's value is **greater than or equal to** the reference value.
- `<lv_obj-bind_state_if_lt>` — Set a state if the subject's value is **less than** the reference value.
- `<lv_obj-bind_state_if_le>` — Set a state if the subject's value is **less than or equal to** the reference value.

Note: The `lv_obj-` prefix can be omitted. For example, you can simply write `<bind_state_if_gt>` instead.

8.2.6 Translations

Panoramica

The XML translation module allows defining and using translated strings directly within XML files.

It's built on top of *LVGL's translation module*. Check it out to learn more about selecting the active language, retrieving translations, and fallback behavior.

Utilizzo

Example XML translation definition:

```
<translations languages="en de hu">
  <translation tag="dog"       en="The dog"     de="Der Hund"       hu="A␣
↪kutya"/>
  <translation tag="cat"       en="The cat"     de="Die Katze"      hu="A␣
↪cica"/>
  <translation tag="snake"     en="A snake"     de="Eine Schlange"  hu="A␣
↪kígyó"/>
</translations>
```

In the root `<translations>` tag, the `languages` attribute defines the available languages, e.g., `languages="en de hu"`. Language codes are free-form, but ISO-style codes are recommended.

Each `<translation>` defines a `tag`, which acts as the lookup key, and attributes for each language.

Translations may be omitted—:ref:`Fallbacks <xml_translations_fallback>` will be applied when needed.

To register XML translations:

- *`lv_xml_translation_register_from_file`*`("path/to/file.xml")`
- *`lv_xml_translation_register_from_data`*`(xml_string)`

Multiple XML sources can be registered; they will be merged and searched collectively.

Usage in XML

Some widget properties support a `*-translated` suffix to refer to translation tags. Ad esempio:

```
<lv_label text-translated="dog"/>
```

This sets the label's text to the translated string for `"dog"`.

8.3 UI Testing

8.3.1 Panoramica

The XML test module is a powerful and flexible way to define functional UI tests.

Test XML files are similar to components but are wrapped in a `<test>` tag and consist of two main parts:

- **UI Definition**: Use `<styles>`, `<consts>`, and `<view>` to define how the UI should look. This is identical to how `<component>`s are structured.
- **Test Steps**: Encapsulated in a `<steps>` tag, these define the actions and assertions for the test.

Step Types

The following step types are currently supported (with example parameters):

- `<move_to x="10" y="30"/>`: Move the mouse to the specified screen coordinates.
- `<click_at x="10" y="30"/>`: Simulates a click at the specified screen coordinates.
- `<press/>`: Press on the current screen coordinate.
- `<release/>`: Release on the current screen coordinate.
- `<wait ms="100"/>`: Waits for the given number of milliseconds. LVGL continues running, including animations and timers.
- `<freeze ms="100"/>`: Pauses the UI and LVGL's internal time. Useful for visual debugging.
- `<subject_set subject="subject1" value="5>` Set an integer or string subject's value.
- `<subject_compare subject="subject1" value="10">` Compare an integer or strings subject's value against a reference value.
- `<screenshot_compare path="path/to/image.png"/>`: Compares the current screen with a reference image. - If the image doesn't exist, it is created. - If the comparison fails, an image with `_err` suffix is saved for inspection.

Esempio:

```
<test width="300" height="400">
    <view width="100%" height="100%" flex_flow="column">
        <lv_checkbox text="First one!"/>
        <lv_slider />
    </view>

    <steps>
        <screenshot_compare path="imgs/before.png"/>
        <click_at x="32" y="32"/>
        <subject_compare subject="subject2" value="50"/>
        <click_at x="50" y="64"/>
        <subject_set subject="subject2" value="10"/>
        <wait ms="300"/>
        <screenshot_compare path="imgs/after.png"/>
    </steps>
</test>
```

Registering Tests

Tests can be registered in two ways:

- From file: `lv_xml_test_register_from_file("path/to/test.xml", "A:ref_images")`
- From string: `lv_xml_test_register_from_data(xml_data, "A:ref_images")`

The second parameter specifies a prefix for screenshot comparison paths.

To unregister tests, use:

- `lv_xml_test_unregister()`

This is called automatically when registering a new test, so manual calls are usually unnecessary.

Running Tests

All steps

To execute the registered test, use `lv_xml_test_run_all(slowdown)`.

It will clean the screen and create a fresh instance of the `view` to be tested.

The `slowdown` parameter controls playback speed:

- `0`: Maximum speed
- `1`: Real-time speed
- `2`: Half-speed
- `10`: 10× slower
- ...and so on

`lv_xml_test_run_all()` blocks until all steps are completed. It is safe to call it multiple times.

The return value is the number of failed tests.

Step-by-step

It's also possible to run each step separately, one after another.

First, call `lv_xml_test_run_init()` to prepare for executing the steps. It will clean the screen and create a fresh instance of the `view` to be tested.

Get the number of steps using `lv_xml_test_get_step_count()` and then call `lv_xml_test_run_next(slowdown)` as many times.

`lv_xml_test_run_next()` returns `true` if the given step passed, or `false` if it failed.

Finally, call `lv_xml_test_run_stop()` to clean up and exit testing mode.

Getting the Test Results

Currently, only `screenshot_compare` steps can fail. However, the result of each step can be queried:

```
uint32_t step_cnt = lv_xml_test_get_step_count();
for(uint32_t i = 0; i < step_cnt; i++) {
    if(lv_xml_test_get_step_type(i) == LV_XML_TEST_STEP_TYPE_SCREENSHOT_
↪COMPARE) {
        if(lv_xml_test_get_status(i))
            printf("Step %d passed\n", i);
        else
            printf("Step %d failed\n", i);
    } else {
        printf("Step %d is not a screenshot comparison\n", i);
    }
}
```

8.3.2 API

lv_test_display.h

lv_test_helpers.h

lv_test_indev.h

lv_test_indev_gesture.h

lv_test_private.h

lv_test_screenshot_compare.h

8.4 Licenza

Version 1.0 – June 2025

Copyright (c) 2025 LVGL LLC

8.4.1 1. Panoramica

The LVGL XML Specification (the "Specification") defines an XML-based language for describing user interfaces within the LVGL ecosystem.

This Specification is protected by copyright. It is provided with limited rights of use as outlined below.

The purpose of this license is to ensure that the Specification remains freely usable for application developers working with LVGL, while restricting its use in third-party UI editors or design tools. This protects the long-term integrity and sustainability of the LVGL ecosystem by preventing unlicensed tools from reusing or replicating the Specification in ways that compete with official LVGL offerings.

8.4.2 2. Permitted Usage

You are permitted to:

- Use this Specification in **any firmware or embedded application** built on top of the official LVGL library (MIT-licensed).
- Use the LVGL XML loader freely in accordance with its MIT license.
- Use the Specification with the **LVGL UI Editor** and **official LVGL tools such as the CLI or XML loader**.
- Write, load, edit, or generate user interfaces described in the LVGL XML Specification for **internal use** only, including automation or configuration tools used solely inside your organization.
- Create **internal scripts or plugins** that help your own developers use or manage this Specification, as long as these are **not shared outside your organization or made publicly available**.
- Share XML-based UI files within customer firmware or projects that target LVGL-based embedded systems.

8.4.3 3. Restrictions

You are **not permitted** to:

- Create, release, or distribute any **UI editor, visual builder, layout designer, code generator, or tool** that reads, writes, or interprets XML files using this Specification — **whether commercial, open-source, or intended for public use, use by customers or partners, or any use outside your own organization** — without written permission from LVGL LLC.
- Implement or extend this Specification in any **software or platform that provides UI creation, editing, or design capabilities similar to those in the LVGL UI Editor**.
- Build public APIs, plugins, converters, or SDKs based on this Specification.
- Share or publish internal tooling built around this Specification.
- Use this Specification into general-purpose design tools or frameworks outside the LVGL ecosystem.

These restrictions apply regardless of whether the software is commercial, free, or open-source.

8.4.4 4. Clarification for LVGL Users

You are **explicitly permitted** to:

- Use XML files with this Specification in your own projects.
- Load XMLs at runtime using LVGL's XML loader.
- Generate XMLs using the LVGL UI Editor.
- Customize XMLs manually or with internal utilities.

- Share LVGL-based projects with XML files as part of a commercial product or firmware.

If you are building embedded software using LVGL, **this license does not affect you**. This license only applies to **UI development tools** that **read, generate, or process XML files using this Specification for UI development purposes** in a public or commercial manner.

You may create **internal tools** to aid development using the Specification, but these must not be published, sold, or shared externally without a license.

8.4.5 5. Trademarks

This license does not grant any rights to use the LVGL name, logo, or trademarks.

8.4.6 6. Licensing and Permissions

To request permission for broader use or commercial licensing (e.g. using this Specification in your own UI tool), contact:

lvgl@lvgl.io

All rights not expressly granted herein are reserved by LVGL LLC.

CAPITOLO 9

Moduli Ausiliari

9.1 Esplora File

`lv_file_explorer` fornisce un'interfaccia utente che consente all'utente finale di esplorare il contenuto di un file system. La sua area principale è chiamata "Browsing Area" [Area di Navigazione] e fornisce l'elenco dei file contenuti nella directory attualmente visualizzata.

Se abilitato, sulla sinistra è presente anche un pannello di "Accesso rapido", che offre un modo comodo per raggiungere le parti del file system a cui si accede di frequente. Available "Quick-Access" destinations are:

- File System,
- HOME,
- Video,
- Pictures,
- Music, and
- Documents.

You specify what paths these lead to during `lv_file_explorer`'s initialization.

`lv_file_explorer` only provides the file browsing and events caused by user activity (e.g. clicking a file), but does not provide the actual file operations. Client code must hook various events and decide what to do when they are emitted (e.g. a click or double-click on a file). The actions taken might to open the file, display it, send it to some other part of the application, etc.. `lv_file_explorer` passes the full path and name of file that was clicked to the event callback functions. What happens next is up to the application designer.

`lv_file_explorer` uses the *Table (lv_table)* Widget for the "Browsing Area", and the *List (lv_list)* Widget for the "Quick-Access" panel when it is enabled. Thus, *`LV_USE_TABLE`* macro must be set to a non-zero value in `lv_conf.h` in order to use `lv_file_explorer`, and and *`LV_USE_LIST`* must be set to a non-zero value to use the "Quick-Access" panel.

> **Nota**
>
> In order to use File Explorer, *File System (lv_fs_drv)* has to be set up and know about all the drive letters you use when passing paths to File System (described below).

9.1.1 Prerequisiti

If you haven't already done so, you will need to learn about the LVGL *File System abstraction*, since it must be set up and be functional for File Explorer to work.

9.1.2 Utilizzo

Set `LV_USE_FILE_EXPLORER` to a non-zero value in `lv_conf.h`.

First use `lv_file_explorer_create(lv_screen_active())` to create a File Explorer. The default size is the screen size. After that, you can customize the style like any Widget.

The size of the `current_path` buffer is set by `LV_FILE_EXPLORER_PATH_MAX_LEN` in `lv_conf.h`.

The object hierarchy of a freshly-created File Explorer looks like this:

- `File Explorer`: occupies full area of parent Widget, typically a Screen (Flex-Flow COLUMN)
  - `Container`: occupies full area of File Explorer (Flex grow 1)
    - `Quick-Access Panel`:
      - `Device List`: grows to accommodate children
      - `File System`: button
      - `Places List`: grows to accommodate children
      - `HOME`: button
      - `Video`: button
      - `Pictures`: button
      - `Music`: button
      - `Documents`: button
    - `Browser Panel`:
      - `Header`: 14% of `Browser Panel` height
      - `Current Path`: label
      - `File Table`: with 1 column, 86% of `Browser Panel` height
  - Campi:
    - `home_dir` = NULL
    - `video_dir` = NULL
    - `pictures_dir` = NULL
    - `music_dir` = NULL
    - `docs_dir` = NULL
    - `fs_dir` = NULL
    - `current_path` = [empty buffer]
    - `sel_fn` (selected file)
    - `sort` (default `LV_EXPLORER_SORT_NONE`)

Accessing the Parts

This list of functions provides access to the parts shown in diagram above:

- `lv_file_explorer_get_selected_file_name(explorer)` (pointer to NUL-terminated string containing file-path user selected; typically used inside an `LV_EVENT_CLICKED` event)

- `lv_file_explorer_get_current_path(explorer)` (pointer to `current_path` `char` buffer)
- `lv_file_explorer_get_file_table(explorer)` (pointer to `File Table` *Table (lv_table)* Widget)
- `lv_file_explorer_get_header(explorer)` (pointer to `Header` *Base Widget (lv_obj)* Widget)
- `lv_file_explorer_get_path_label(explorer)` (pointer to `Current Path Label` *Label (lv_label)* Widget)
- `lv_file_explorer_get_quick_access_area(explorer)` (pointer to `Quick-Access Panel` *Base Widget (lv_obj)*)
- `lv_file_explorer_get_places_list(explorer)` (pointer to `Places List` *List (lv_list)* Widget)
- `lv_file_explorer_get_device_list(explorer)` (pointer to `Device List` *List (lv_list)* Widget)

Quick-Access Panel

The `Quick-Access Panel` behaves like a typical navigation panel and appears on the left, while the `Browser Panel` appears on the right

This panel is optional. If you set `LV_FILE_EXPLORER_QUICK_ACCESS` to `0` in `lv_conf.h`, the `Quick-Access Panel` will not be created. This saves only a little bit of memory.

Soon after the File Explorer is created, you typically use `lv_file_explorer_set_quick_access_path(explorer, LV_EXPLORER_XXX_DIR, "path")` to set the path that will be navigated to when the buttons in the `Quick-Access Panel` are clicked, which is currently a fixed list. The corresponding values you will need to pass as the 2nd argument are the following:

- `LV_EXPLORER_HOME_DIR`
- `LV_EXPLORER_MUSIC_DIR`
- `LV_EXPLORER_PICTURES_DIR`
- `LV_EXPLORER_VIDEO_DIR`
- `LV_EXPLORER_DOCS_DIR`
- `LV_EXPLORER_FS_DIR`

Sort

You can use `lv_file_explorer_set_sort(explorer, LV_EXPLORER_SORT_XX)` to set the sorting method.

These are the possible sorting methods:

- `LV_EXPLORER_SORT_NONE` (default)
- `LV_EXPLORER_SORT_KIND`

`lv_file_explorer_get_sort(explorer)` returns the current sorting method.

9.1.3 Eventi

- `LV_EVENT_READY` Sent when a directory is opened, which can happen:
  - when the File Explorer is initially opened,
  - after a user clicks on a `Quick-Access Panel` navigation button, and
  - after the user clicks on a directory displayed in the `Browser Panel`.

  You can use it to, for example, customize the file sort.

- *LV_EVENT_VALUE_CHANGED* Sent once when any item (file) in the `Browser Panel`'s file list is clicked.
- *LV_EVENT_CLICKED* Sent twice when an item in the `Browser Panel` is clicked: once as a result of the input-device *LV_EVENT_RELEASED* event and a second as a result of the input device *LV_EVENT_CLICKED* event. This applies to files, directories, and the "< Back" item in the `Browser Panel`.

In these events you can use *lv_file_explorer_get_current_path()* to get the current path and *lv_file_explorer_get_selected_file_name()* to get the name of the currently selected file in the event processing function. Ad esempio:

```
static void file_explorer_event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target(e);

    if(code == LV_EVENT_VALUE_CHANGED) {
        char * cur_path =  lv_file_explorer_get_current_path(widget);
        char * sel_fn = lv_file_explorer_get_selected_file_name(widget);
        LV_LOG_USER("%s%s", cur_path, sel_fn);
    }
}
```

You can also save the obtained **path** and **file** name into an array through functions such as `strcpy()` and `strcat()` for later use.

9.1.4 Esempio

Simple File Explorer

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

#if LV_USE_TABLE && LV_USE_FILE_EXPLORER && (LV_USE_FS_STDIO || LV_USE_FS_
↪POSIX || LV_USE_FS_WIN32 || LV_USE_FS_FATFS) && LV_BUILD_EXAMPLES

#include <stdlib.h>
#include <string.h>

static void file_explorer_event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);

    if(code == LV_EVENT_VALUE_CHANGED) {
        const char * cur_path =  lv_file_explorer_get_current_path(obj);
        const char * sel_fn = lv_file_explorer_get_selected_file_name(obj);
        LV_LOG_USER("%s%s", cur_path, sel_fn);
    }
}

void lv_example_file_explorer_1(void)
{
    lv_obj_t * file_explorer = lv_file_explorer_create(lv_screen_active());
    lv_file_explorer_set_sort(file_explorer, LV_EXPLORER_SORT_KIND);

#if LV_USE_FS_WIN32
    /* Note to Windows users:  the initial "C:" on these paths corresponds␣
↪to
```

(continues on next page)

(continua dalla pagina precedente)

```
     * the value of `LV_FS_WIN32_LETTER` in `lv_conf.h`, and should not be
     * confused with the Windows/DOS drive letter.  It is an identifier␣
↪that
     * is used to enable LVGL to look up the appropriate driver from a␣
↪list of
     * registered file-system drivers.  `lv_fs_win32_init()` happens to␣
↪use the
     * identifier letter 'C' so "C:" is the driver-identifier-prefix used␣
↪here.
     * The "C:" following that is indeed the Windows/DOS drive letter and␣
↪is
     * part of the actual path that gets passed to the OS-level functions.
     *
     * See https://docs.lvgl.io/master/details/main-components/fs.html for␣
↪details.
     * File Explorer uses `lv_fs` internally, thus the required prefix in␣
↪path strings.
     */
    lv_file_explorer_open_dir(file_explorer, "C:C:/");
#if LV_FILE_EXPLORER_QUICK_ACCESS
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_HOME_
↪DIR, "C:C:/Users/Public/Desktop");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_
↪VIDEO_DIR, "C:C:/Users/Public/Videos");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_
↪PICTURES_DIR, "C:C:/Users/Public/Pictures");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_
↪MUSIC_DIR, "C:C:/Users/Public/Music");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_DOCS_
↪DIR, "C:C:/Users/Public/Documents");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_FS_
↪DIR, "C:C:/");
#endif

#else
    /* linux */
    lv_file_explorer_open_dir(file_explorer, "A:/");

#if LV_FILE_EXPLORER_QUICK_ACCESS
    const char * envvar = "HOME";
    char home_dir[LV_FS_MAX_PATH_LENGTH];
    strcpy(home_dir, "A:");
    /* get the user's home directory from the HOME environment variable*/
    strcat(home_dir, getenv(envvar));
    LV_LOG_USER("home_dir: %s\n", home_dir);
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_HOME_
↪DIR, home_dir);
    char video_dir[LV_FS_MAX_PATH_LENGTH];
    strcpy(video_dir, home_dir);
    strcat(video_dir, "/Videos");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_
↪VIDEO_DIR, video_dir);
    char picture_dir[LV_FS_MAX_PATH_LENGTH];
    strcpy(picture_dir, home_dir);
    strcat(picture_dir, "/Pictures");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_
```

(continues on next page)

(continua dalla pagina precedente)

```
↪PICTURES_DIR, picture_dir);
    char music_dir[LV_FS_MAX_PATH_LENGTH];
    strcpy(music_dir, home_dir);
    strcat(music_dir, "/Music");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_
↪MUSIC_DIR, music_dir);
    char document_dir[LV_FS_MAX_PATH_LENGTH];
    strcpy(document_dir, home_dir);
    strcat(document_dir, "/Documents");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_DOCS_
↪DIR, document_dir);

    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_FS_
↪DIR, "A:/");
#endif
#endif

    lv_obj_add_event_cb(file_explorer, file_explorer_event_handler, LV_
↪EVENT_ALL, NULL);
}

#endif
```

Control File Explorer

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

#if LV_USE_TABLE && LV_USE_DROPDOWN && LV_USE_FILE_EXPLORER && (LV_USE_FS_
↪STDIO || LV_USE_FS_POSIX || LV_USE_FS_WIN32 || LV_USE_FS_FATFS) && LV_
↪BUILD_EXAMPLES

#include <stdlib.h>
#include <string.h>

static void file_explorer_event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);

    if(code == LV_EVENT_VALUE_CHANGED) {
        const char * cur_path =  lv_file_explorer_get_current_path(obj);
        const char * sel_fn = lv_file_explorer_get_selected_file_name(obj);

        LV_LOG_USER("%s%s", cur_path, sel_fn);
    }
}

#if LV_FILE_EXPLORER_QUICK_ACCESS
static void btn_event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * btn = lv_event_get_target_obj(e);
    lv_obj_t * file_explorer = (lv_obj_t *) lv_event_get_user_data(e);

    if(code == LV_EVENT_VALUE_CHANGED) {
```

(continues on next page)

(continua dalla pagina precedente)

```
        if(lv_obj_has_state(btn, LV_STATE_CHECKED))
            lv_obj_add_flag(file_explorer, LV_OBJ_FLAG_HIDDEN);
        else
            lv_obj_remove_flag(file_explorer, LV_OBJ_FLAG_HIDDEN);
    }
}

static void dd_event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * dd = lv_event_get_target_obj(e);
    lv_obj_t * fe_quick_access_obj = (lv_obj_t *) lv_event_get_user_
↪data(e);

    if(code == LV_EVENT_VALUE_CHANGED) {
        char buf[32];
        lv_dropdown_get_selected_str(dd, buf, sizeof(buf));
        if(strcmp(buf, "NONE") == 0) {
            lv_file_explorer_set_sort(fe_quick_access_obj, LV_EXPLORER_
↪SORT_NONE);
        }
        else if(strcmp(buf, "KIND") == 0) {
            lv_file_explorer_set_sort(fe_quick_access_obj, LV_EXPLORER_
↪SORT_KIND);
        }
    }
}
#endif

void lv_example_file_explorer_2(void)
{
    lv_obj_t * file_explorer = lv_file_explorer_create(lv_screen_active());

#if LV_USE_FS_WIN32
    /* Note to Windows users:  the initial "C:" on these paths corresponds␣
↪to
     * the value of `LV_FS_WIN32_LETTER` in `lv_conf.h`, and should not be
     * confused with the Windows/DOS drive letter.  It is an identifier␣
↪that
     * is used to enable LVGL to look up the appropriate driver from a␣
↪list of
     * registered file-system drivers.  `lv_fs_win32_init()` happens to␣
↪use the
     * identifier letter 'C' so "C:" is the driver-identifier-prefix used␣
↪here.
     * The "C:" following that is indeed the Windows/DOS drive letter and␣
↪is
     * part of the actual path that gets passed to the OS-level functions.
     *
     * See https://docs.lvgl.io/master/details/main-components/fs.html for␣
↪details.
     * File Explorer uses `lv_fs` internally, thus the required prefix in␣
↪path strings.
     */
    lv_file_explorer_open_dir(file_explorer, "C:C:/");
#if LV_FILE_EXPLORER_QUICK_ACCESS
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_HOME_
↪DIR, "C:C:/Users/Public/Desktop");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_
↪VIDEO_DIR, "C:C:/Users/Public/Videos");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_
↪PICTURES_DIR, "C:C:/Users/Public/Pictures");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_
↪MUSIC_DIR, "C:C:/Users/Public/Music");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_DOCS_
↪DIR, "C:C:/Users/Public/Documents");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_FS_
↪DIR, "C:C:/");
#endif

#else
    /* linux */
    lv_file_explorer_open_dir(file_explorer, "A:/");
#if LV_FILE_EXPLORER_QUICK_ACCESS
    const char * envvar = "HOME";
    char home_dir[LV_FS_MAX_PATH_LENGTH];
    strcpy(home_dir, "A:");
    /* get the user's home directory from the HOME environment variable*/
    strcat(home_dir, getenv(envvar));
    LV_LOG_USER("home_dir: %s\n", home_dir);
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_HOME_
↪DIR, home_dir);
    char video_dir[LV_FS_MAX_PATH_LENGTH];
    strcpy(video_dir, home_dir);
    strcat(video_dir, "/Videos");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_
↪VIDEO_DIR, video_dir);
    char picture_dir[LV_FS_MAX_PATH_LENGTH];
    strcpy(picture_dir, home_dir);
    strcat(picture_dir, "/Pictures");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_
↪PICTURES_DIR, picture_dir);
    char music_dir[LV_FS_MAX_PATH_LENGTH];
    strcpy(music_dir, home_dir);
    strcat(music_dir, "/Music");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_
↪MUSIC_DIR, music_dir);
    char document_dir[LV_FS_MAX_PATH_LENGTH];
    strcpy(document_dir, home_dir);
    strcat(document_dir, "/Documents");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_DOCS_
↪DIR, document_dir);

    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_FS_
↪DIR, "A:/");
#endif
#endif

    lv_obj_add_event_cb(file_explorer, file_explorer_event_handler, LV_
↪EVENT_ALL, NULL);

#if LV_FILE_EXPLORER_QUICK_ACCESS
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Quick access status control button*/
    lv_obj_t * fe_quick_access_obj = lv_file_explorer_get_quick_access_
↪area(file_explorer);
    lv_obj_t * fe_header_obj = lv_file_explorer_get_header(file_explorer);
    lv_obj_t * btn = lv_button_create(fe_header_obj);
    lv_obj_set_style_radius(btn, 2, 0);
    lv_obj_set_style_pad_all(btn, 4, 0);
    lv_obj_align(btn, LV_ALIGN_LEFT_MID, 0, 0);
    lv_obj_add_flag(btn, LV_OBJ_FLAG_CHECKABLE);

    lv_obj_t * label = lv_label_create(btn);
    lv_label_set_text(label, LV_SYMBOL_LIST);
    lv_obj_center(label);

    lv_obj_add_event_cb(btn, btn_event_handler, LV_EVENT_VALUE_CHANGED, fe_
↪quick_access_obj);

    /*Sort control*/
    static const char * opts = "NONE\n"
                               "KIND";

    lv_obj_t * dd = lv_dropdown_create(fe_header_obj);
    lv_obj_set_style_radius(dd, 4, 0);
    lv_obj_set_style_pad_all(dd, 0, 0);
    lv_obj_set_size(dd, LV_PCT(30), LV_SIZE_CONTENT);
    lv_dropdown_set_options_static(dd, opts);
    lv_obj_align(dd, LV_ALIGN_RIGHT_MID, 0, 0);

    lv_obj_add_event_cb(dd, dd_event_handler, LV_EVENT_VALUE_CHANGED, file_
↪explorer);
#endif
}

#endif
```

Custom sort

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

#if LV_USE_TABLE && LV_USE_FILE_EXPLORER && (LV_USE_FS_STDIO || LV_USE_FS_
↪POSIX || LV_USE_FS_WIN32 || LV_USE_FS_FATFS) && LV_BUILD_EXAMPLES

#include <stdlib.h>
#include <string.h>

static void exch_table_item(lv_obj_t * tb, int16_t i, int16_t j)
{
    const char * tmp;
    tmp = lv_table_get_cell_value(tb, i, 0);
    lv_table_set_cell_value(tb, 0, 2, tmp);
    lv_table_set_cell_value(tb, i, 0, lv_table_get_cell_value(tb, j, 0));
    lv_table_set_cell_value(tb, j, 0, lv_table_get_cell_value(tb, 0, 2));

    tmp = lv_table_get_cell_value(tb, i, 1);
    lv_table_set_cell_value(tb, 0, 2, tmp);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_table_set_cell_value(tb, i, 1, lv_table_get_cell_value(tb, j, 1));
    lv_table_set_cell_value(tb, j, 1, lv_table_get_cell_value(tb, 0, 2));
}

/*Quick sort 3 way*/
static void sort_by_file_kind(lv_obj_t * tb, int16_t lo, int16_t hi)
{
    if(lo >= hi) return;

    int16_t lt = lo;
    int16_t i = lo + 1;
    int16_t gt = hi;
    const char * v = lv_table_get_cell_value(tb, lo, 1);
    while(i <= gt) {
        if(strcmp(lv_table_get_cell_value(tb, i, 1), v) < 0)
            exch_table_item(tb, lt++, i++);
        else if(strcmp(lv_table_get_cell_value(tb, i, 1), v) > 0)
            exch_table_item(tb, i, gt--);
        else
            i++;
    }

    sort_by_file_kind(tb, lo, lt - 1);
    sort_by_file_kind(tb, gt + 1, hi);
}

static void file_explorer_event_handler(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);

    if(code == LV_EVENT_VALUE_CHANGED) {
        const char * cur_path =  lv_file_explorer_get_current_path(obj);
        const char * sel_fn = lv_file_explorer_get_selected_file_name(obj);

        LV_LOG_USER("%s%s", cur_path, sel_fn);
    }
    else if(code == LV_EVENT_READY) {
        lv_obj_t * tb = lv_file_explorer_get_file_table(obj);
        uint16_t sum = lv_table_get_row_count(tb);

        sort_by_file_kind(tb, 0, (sum - 1));
    }
}

void lv_example_file_explorer_3(void)
{
    lv_obj_t * file_explorer = lv_file_explorer_create(lv_screen_active());
    /*Before custom sort, please set the default sorting to NONE. The␣
↪default is NONE.*/
    lv_file_explorer_set_sort(file_explorer, LV_EXPLORER_SORT_NONE);

#if LV_USE_FS_WIN32
    /* Note to Windows users:  the initial "C:" on these paths corresponds␣
↪to
     * the value of `LV_FS_WIN32_LETTER` in `lv_conf.h`, and should not be
```

(continues on next page)

(continua dalla pagina precedente)

```
     * confused with the Windows/DOS drive letter.  It is an identifier␣
↪that
     * is used to enable LVGL to look up the appropriate driver from a␣
↪list of
     * registered file-system drivers.  `lv_fs_win32_init()` happens to␣
↪use the
     * identifier letter 'C' so "C:" is the driver-identifier-prefix used␣
↪here.
     * The "C:" following that is indeed the Windows/DOS drive letter and␣
↪is
     * part of the actual path that gets passed to the OS-level functions.
     *
     * See https://docs.lvgl.io/master/details/main-components/fs.html for␣
↪details.
     * File Explorer uses `lv_fs` internally, thus the required prefix in␣
↪path strings.
     */
    lv_file_explorer_open_dir(file_explorer, "C:C:/");
#if LV_FILE_EXPLORER_QUICK_ACCESS
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_HOME_
↪DIR, "C:C:/Users/Public/Desktop");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_
↪VIDEO_DIR, "C:C:/Users/Public/Videos");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_
↪PICTURES_DIR, "C:C:/Users/Public/Pictures");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_
↪MUSIC_DIR, "C:C:/Users/Public/Music");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_DOCS_
↪DIR, "C:C:/Users/Public/Documents");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_FS_
↪DIR, "C:C:/");
#endif

#else
    /* linux */
    lv_file_explorer_open_dir(file_explorer, "A:/");
#if LV_FILE_EXPLORER_QUICK_ACCESS
    const char * envvar = "HOME";
    char home_dir[LV_FS_MAX_PATH_LENGTH];
    strcpy(home_dir, "A:");
    /* get the user's home directory from the HOME environment variable*/
    strcat(home_dir, getenv(envvar));
    LV_LOG_USER("home_dir: %s\n", home_dir);
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_HOME_
↪DIR, home_dir);
    char video_dir[LV_FS_MAX_PATH_LENGTH];
    strcpy(video_dir, home_dir);
    strcat(video_dir, "/Videos");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_
↪VIDEO_DIR, video_dir);
    char picture_dir[LV_FS_MAX_PATH_LENGTH];
    strcpy(picture_dir, home_dir);
    strcat(picture_dir, "/Pictures");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_
↪PICTURES_DIR, picture_dir);
    char music_dir[LV_FS_MAX_PATH_LENGTH];
```

(continues on next page)

(continua dalla pagina precedente)

```
    strcpy(music_dir, home_dir);
    strcat(music_dir, "/Music");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_
↪MUSIC_DIR, music_dir);
    char document_dir[LV_FS_MAX_PATH_LENGTH];
    strcpy(document_dir, home_dir);
    strcat(document_dir, "/Documents");
    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_DOCS_
↪DIR, document_dir);

    lv_file_explorer_set_quick_access_path(file_explorer, LV_EXPLORER_FS_
↪DIR, "A:/");
#endif
#endif

    lv_obj_add_event_cb(file_explorer, file_explorer_event_handler, LV_
↪EVENT_ALL, NULL);
}

#endif
```

9.1.5 API

lv_file_explorer.h

lv_file_explorer_private.h

9.2 Font Manager

The font manager is a secondary encapsulation of the LVGL font engine, which facilitates the use and management of fonts for applications. The font management functions currently provided include:

- Font resource reference counting (reduces repeated creation of font resources).
- Font resource concatenation (font fallback).
- Font resource recycling mechanism (buffers recently deleted font resources to reduce the time overhead of repeated creation).

9.2.1 Utilizzo

Enable FreeType and Font Manager in `lv_conf.h` by setting the *LV_USE_FONT_MANAGER* macros to non-zero values, and configure `LV_FONT_MANAGER_NAME_MAX_LEN` to set the maximum length of the font name.

Initialize Font Manager

Use *lv_font_manager_create()* to create a font manager, where the `recycle_cache_size()` parameter is used to set the number of font recycling caches, which can improve font creation efficiency.

Use *lv_font_manager_add_src_static()* to add a mapping between font names and font resources to tell the font manager how to access the font resources. Note that if the font resource description structure is not statically allocated (for example, allocated from a local variable), use *lv_font_manager_add_src()* to add the resource. This function will copy the contents of the structure itself.

Use *lv_font_manager_remove_src()* to remove the font resource mapping.

It should be noted that the `src` parameter must strictly correspond to `class_p`. `class_p` will affect the way the font manager interprets src. If an incompatible parameter is passed, the program may fail. For currently supported font classes, please refer to the example code.

```
static lv_font_manager_t * g_font_manager = NULL;

void font_manager_init(void)
{
   /* Create font manager, with 8 fonts recycling buffers */
   g_font_manager = lv_font_manager_create(8);

   /* Add font path mapping to font manager */
   lv_font_manager_add_src_static(g_font_manager, "Lato-Regular", "./lvgl/
↪examples/libs/freetype/Lato-Regular.ttf", &lv_freetype_font_class);

   char path[] = "/path/to/myfont.ttf";
   lv_font_manager_add_src(g_font_manager, "MyFont", path, &lv_freetype_
↪font_class);
}
```

Create Font from Font Manager

The parameters will be passed to the font creation function of the font backend, such as *lv_freetype_font_create()* and `lv_tiny_ttf_create_file()`. The font backend will select the supported parameters by itself and ignore the unsupported parameters.

The `font_family` parameter can be filled with the names of multiple fonts (separated by `,`) to achieve font concatenation (when the corresponding glyph is not found in a font file, it will automatically search from the next concatenated font).

```
static lv_font_t * g_font = NULL;

/* Create font from font manager */
lv_font_t * g_font = lv_font_manager_create_font(g_font_manager,
                                                 "Lato-Regular,MyFont",
                                                 LV_FREETYPE_FONT_RENDER_
↪MODE_BITMAP,
                                                 24,
                                                 LV_FREETYPE_FONT_STYLE_
↪NORMAL,
                                                 LV_FONT_KERNING_NONE);

/* Handle error case */
if(g_font == NULL) {
   g_font = (lv_font_t *)LV_FONT_DEFAULT;
}

/* Create label with the font */
lv_obj_t * label = lv_label_create(lv_screen_active());
lv_obj_set_style_text_font(label, g_font, 0);
lv_label_set_text(label, "Hello World!");
```

Delete Font

Use *lv_font_manager_delete_font()* to delete the font when it is no longer needed. The font manager will mark the font resource as a recyclable font so that it has the possibility of being more quickly created next time.

Note that you need to delete any Widgets that used the font first, and then delete the font to avoid accessing wild pointers.

```
/* Delete label and font */
lv_obj_delete(label);
```

(continues on next page)

(continua dalla pagina precedente)

```
lv_font_manager_delete_font(g_font_manager, g_font);
g_font = NULL;
```

Delete Font Manager

Use *lv_font_manager_delete()* to destroy the entire font manager. It should be noted that before destruction, it is necessary to ensure that the application has deleted all fonts using *lv_font_manager_delete_font()*. The font manager will check the reference status of all allocated fonts. If there are still fonts being referenced, the font manager will fail to be destroyed and the function will return false.

9.2.2 Esempio

Create FreeType Font

```
#include "../../lv_examples.h"
#if LV_USE_FONT_MANAGER && LV_USE_FREETYPE && LV_BUILD_EXAMPLES

#if LV_FREETYPE_USE_LVGL_PORT
    #define PATH_PREFIX "A:"
#else
    #define PATH_PREFIX "./"
#endif

static lv_font_manager_t * g_font_manager = NULL;

void lv_example_font_manager_1(void)
{
    /* Create font manager, with 8 fonts recycling buffers */
    g_font_manager = lv_font_manager_create(8);

    /* Add font path mapping to font manager */
    lv_font_manager_add_src_static(g_font_manager,
                                   "Lato-Regular",
                                   PATH_PREFIX "lvgl/examples/libs/
↪freetype/Lato-Regular.ttf",
                                   &lv_freetype_font_class);

    /* Create font from font manager */
    lv_font_t * font = lv_font_manager_create_font(g_font_manager,
                                                   "Lato-Regular",
                                                   LV_FREETYPE_FONT_RENDER_
↪MODE_BITMAP,
                                                   24,
                                                   LV_FREETYPE_FONT_STYLE_
↪NORMAL,
                                                   LV_FONT_KERNING_NONE);

    if(!font) {
        LV_LOG_ERROR("Could not create font");
        return;
    }

    /* Create label with the font */
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_set_style_text_font(label, font, 0);
    lv_label_set_text(label, "Hello Font Manager!");
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_center(label);
}

#else

void lv_example_font_manager_1(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Font Manager or FreeType is not enabled");
    lv_obj_center(label);
}

#endif
```

Create Font Family

```
#include "../../lv_examples.h"
#if LV_USE_FONT_MANAGER && LV_BUILD_EXAMPLES

#if defined(LV_FREETYPE_USE_LVGL_PORT) && LV_FREETYPE_USE_LVGL_PORT
    #define PATH_PREFIX "A:"
#else
    #define PATH_PREFIX "./"
#endif

static lv_font_manager_t * g_font_manager = NULL;

void lv_example_font_manager_2(void)
{
    /* Create font manager, with 8 fonts recycling buffers */
    g_font_manager = lv_font_manager_create(8);

#if LV_FONT_MONTSERRAT_14 && LV_FONT_MONTSERRAT_32

    /* Register built-in font sources */
    static lv_builtin_font_src_t builtin_font_src[3] = { 0 };
    builtin_font_src[0].font_p = &lv_font_montserrat_14;
    builtin_font_src[0].size = 14;
    builtin_font_src[1].font_p = &lv_font_montserrat_32;
    builtin_font_src[1].size = 32;

    /* IMPORTANT! Marking the end of the array */
    builtin_font_src[2].font_p = NULL;
    builtin_font_src[2].size = 0;

    lv_font_manager_add_src_static(g_font_manager,
                                   "Montserrat",
                                   builtin_font_src,
                                   &lv_builtin_font_class);
#endif

#if LV_USE_FREETYPE
    /* Register FreeType font source */
    lv_font_manager_add_src_static(g_font_manager,
                                   "NotoColorEmoji",
                                   PATH_PREFIX "lvgl/examples/libs/
```

(continues on next page)

(continua dalla pagina precedente)

```
↪freetype/NotoColorEmoji-32.subset.ttf",
                                    &lv_freetype_font_class);
#endif

#if LV_USE_TINY_TTF && LV_TINY_TTF_FILE_SUPPORT
    /* Register TinyTTF font source */
    static const lv_tiny_ttf_font_src_t tiny_ttf_font_src = {
        .path = "A:lvgl/examples/libs/tiny_ttf/Ubuntu-Medium.ttf",
        .data = NULL,
        .data_size = 0,
        .cache_size = 0,
    };

    lv_font_manager_add_src_static(g_font_manager,
                                    "Ubuntu-Medium",
                                    &tiny_ttf_font_src,
                                    &lv_tiny_ttf_font_class);
#endif

    /* Create font from font manager */
    lv_font_t * font = lv_font_manager_create_font(g_font_manager,
                                                    "Ubuntu-Medium,
↪NotoColorEmoji,Montserrat",
                                                    LV_FREETYPE_FONT_RENDER_
↪MODE_BITMAP,
                                                    32,
                                                    LV_FREETYPE_FONT_STYLE_
↪NORMAL,
                                                    LV_FONT_KERNING_NONE);

    if(!font) {
        LV_LOG_ERROR("Could not create font");
        return;
    }

    /* Create label with the font */
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_set_style_text_font(label, font, 0);
    lv_label_set_text(label, "Hello Font Manager! □ " LV_SYMBOL_OK);
    lv_obj_center(label);
}

#else

void lv_example_font_manager_2(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Font Manager is not enabled");
    lv_obj_center(label);
}

#endif
```

Create Custom Image Font

```
#include "../../lv_examples.h"
#if LV_USE_FONT_MANAGER && LV_USE_FREETYPE && LV_USE_IMGFONT && LV_BUILD_
↪EXAMPLES

#if LV_FREETYPE_USE_LVGL_PORT
    #define PATH_PREFIX "A:"
#else
    #define PATH_PREFIX "./"
#endif

typedef struct {
    uint16_t match_size_min;
    uint16_t match_size_max;
    lv_imgfont_get_path_cb_t get_path_cb;
} imgfont_src_t;

static lv_font_manager_t * g_font_manager = NULL;

static const void * get_imgfont_path(const lv_font_t * font, uint32_t␣
↪unicode, uint32_t unicode_next,
                                     int32_t * offset_y, void * user_data)
{
    LV_UNUSED(font);
    LV_UNUSED(unicode_next);
    LV_UNUSED(offset_y);
    LV_UNUSED(user_data);

    LV_IMAGE_DECLARE(emoji_F617);

    if(unicode == 0xF617) {
        return &emoji_F617;
    }

    return NULL;
}

static lv_font_t * imgfont_create_cb(const lv_font_info_t * info, const␣
↪void * src)
{
    const imgfont_src_t * imgfont_src = (imgfont_src_t *) src;

    if(info->size < imgfont_src->match_size_min
       || info->size > imgfont_src->match_size_max) {
        return NULL;
    }

    return lv_imgfont_create(info->size, imgfont_src->get_path_cb, NULL);
}

static void imgfont_delete_cb(lv_font_t * font)
{
    lv_imgfont_destroy(font);
}

static void * imgfont_dup_src_cb(const void * src)
```

(continues on next page)

(continua dalla pagina precedente)

```
{
    imgfont_src_t * imgfont_src = (imgfont_src_t *) lv_
↪malloc(sizeof(imgfont_src_t));
    LV_ASSERT_MALLOC(imgfont_src);
    lv_memcpy(imgfont_src, src, sizeof(imgfont_src_t));
    return imgfont_src;
}

static void imgfont_free_src_cb(void * src)
{
    lv_free(src);
}

static const lv_font_class_t imgfont_class = {
    .create_cb = imgfont_create_cb,
    .delete_cb = imgfont_delete_cb,
    .dup_src_cb = imgfont_dup_src_cb,
    .free_src_cb = imgfont_free_src_cb,
};

void lv_example_font_manager_3(void)
{
    /* Create font manager, with 8 fonts recycling buffers */
    g_font_manager = lv_font_manager_create(8);

    /* Add font path mapping to font manager */
    lv_font_manager_add_src_static(g_font_manager,
                                   "Lato-Regular",
                                   PATH_PREFIX "lvgl/examples/libs/
↪freetype/Lato-Regular.ttf",
                                   &lv_freetype_font_class);

    /* Add custom imgfont source to font manager */
    static const imgfont_src_t imgfont_src = {
        .match_size_min = 70,
        .match_size_max = 80,
        .get_path_cb = get_imgfont_path,
    };
    lv_font_manager_add_src_static(g_font_manager,
                                   "Emoji",
                                   &imgfont_src,
                                   &imgfont_class);

    /* Create font from font manager */
    lv_font_t * font = lv_font_manager_create_font(g_font_manager,
                                                   "Lato-Regular,Emoji",
                                                   LV_FREETYPE_FONT_RENDER_
↪MODE_BITMAP,
                                                   75,
                                                   LV_FREETYPE_FONT_STYLE_
↪NORMAL,
                                                   LV_FONT_KERNING_NONE);

    if(!font) {
        LV_LOG_ERROR("Could not create font");
        return;
```

(continues on next page)

(continua dalla pagina precedente)

```
    }

    /* Create label with the font */
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_set_style_text_font(label, font, 0);
    lv_label_set_text(label, "Quiet\uF617~");
    lv_obj_center(label);
}

#else

void lv_example_font_manager_3(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Font Manager or Freetype or Imgfont is not␣
↪enabled");
    lv_obj_center(label);
}

#endif
```

9.2.3 API

lv_font_manager.h

lv_font_manager_recycle.h

9.3 Fragment

Fragment is a concept copied from Android.

It represents a reusable portion of your app's UI. A fragment defines and manages its own layout, has its own lifecycle, and can handle its own events. Like Android's Fragment that must be hosted by an activity or another fragment, Fragment in LVGL needs to be hosted by a Widget, or another fragment. The fragment's view hierarchy becomes part of, or attaches to, the host's view hierarchy.

Such concept also has some similarities to UiViewController on iOS.

Fragment Manager is a manager holding references to fragments attached to it, and has an internal stack to achieve forward and backwards navigation. You can use fragment manager to build a navigation stack, or a multi-pane application easily.

9.3.1 Utilizzo

Enable *LV_USE_FRAGMENT* in `lv_conf.h`.

Create Fragment Class

```
struct sample_fragment_t {
    /* IMPORTANT: don't miss this part */
    lv_fragment_t base;
    /* States, object references and data fields for this fragment */
    const char *title;
};

const lv_fragment_class_t sample_cls = {
        /* Initialize something needed */
```

(continues on next page)

(continua dalla pagina precedente)

```
        .constructor_cb = sample_fragment_ctor,
        /* Create view objects */
        .create_obj_cb = sample_fragment_create_obj,
        /* IMPORTANT: size of your fragment struct */
        .instance_size = sizeof(struct sample_fragment_t),
};
```

Use `lv_fragment_manager`

```
/* Create fragment instance, and Widgets will be added to container */
lv_fragment_manager_t *manager = lv_fragment_manager_create(container,␣
↪NULL);
/* Replace current fragment with instance of sample_cls, and init_argument␣
↪is user defined pointer */
lv_fragment_manager_replace(manager, &sample_cls, init_argument);
```

Fragment Based Navigation

```
/* Add one instance into manager stack. View object of current fragment␣
↪will be destroyed,
 * but instances created in class constructor will be kept.
 */
lv_fragment_manager_push(manager, &sample_cls, NULL);

/* Remove the top most fragment from the stack, and bring back previous␣
↪one. */
lv_fragment_manager_pop(manager);
```

9.3.2 Esempio

Basic fragment usage

```
/**
 * @file lv_example_fragment_1.c
 * @brief Basic usage of obj fragment
 */
#include "../../lv_examples.h"

#if LV_USE_FRAGMENT && LV_BUILD_EXAMPLES

static void sample_fragment_ctor(lv_fragment_t * self, void * args);

static lv_obj_t * sample_fragment_create_obj(lv_fragment_t * self, lv_obj_
↪t * parent);

static void sample_container_delete(lv_event_t * e);

static lv_obj_t * root = NULL;

struct sample_fragment_t {
    lv_fragment_t base;
    const char * name;
};

static const lv_fragment_class_t sample_cls = {
```

(continues on next page)

(continua dalla pagina precedente)

```
    .constructor_cb = sample_fragment_ctor,
    .create_obj_cb = sample_fragment_create_obj,
    .instance_size = sizeof(struct sample_fragment_t),
    };

void lv_example_fragment_1(void)
{
    root = lv_obj_create(lv_screen_active());
    lv_obj_set_size(root, LV_PCT(100), LV_PCT(100));
    lv_fragment_manager_t * manager = lv_fragment_manager_create(NULL);
    /* Clean up the fragment manager before objects in containers got␣
↪deleted */
    lv_obj_add_event_cb(root, sample_container_delete, LV_EVENT_DELETE,␣
↪manager);

    lv_fragment_t * fragment = lv_fragment_create(&sample_cls, (void *)
↪"Fragment");
    lv_fragment_manager_replace(manager, fragment, &root);
}

static void sample_fragment_ctor(lv_fragment_t * self, void * args)
{
    ((struct sample_fragment_t *) self)->name = (const char *) args;
}

static lv_obj_t * sample_fragment_create_obj(lv_fragment_t * self, lv_obj_
↪t * parent)
{
    lv_obj_t * label = lv_label_create(parent);
    lv_obj_set_style_bg_opa(label, LV_OPA_COVER, 0);;
    lv_label_set_text_fmt(label, "Hello, %s!", ((struct sample_fragment_t␣
↪*) self)->name);
    return label;
}

static void sample_container_delete(lv_event_t * e)
{
    lv_fragment_manager_t * manager = (lv_fragment_manager_t *) lv_event_
↪get_user_data(e);
    lv_fragment_manager_delete(manager);
}

#endif
```

Stack navigation example

```
/**
 * @file lv_example_fragment_2.c
 * @brief Navigation stack using obj fragment
 */
#include "../../lv_examples.h"

#if LV_USE_FRAGMENT && LV_USE_WIN && LV_USE_GRID && LV_BUILD_EXAMPLES

static void sample_fragment_ctor(lv_fragment_t * self, void * args);
```

(continues on next page)

(continua dalla pagina precedente)

```
static lv_obj_t * sample_fragment_create_obj(lv_fragment_t * self, lv_obj_
↪t * parent);

static void sample_push_click(lv_event_t * e);

static void sample_pop_click(lv_event_t * e);

static void sample_container_delete(lv_event_t * e);

static void sample_fragment_inc_click(lv_event_t * e);

typedef struct sample_fragment_t {
    lv_fragment_t base;
    lv_obj_t * label;
    int depth;
    int counter;
} sample_fragment_t;

static const lv_fragment_class_t sample_cls = {
    .constructor_cb = sample_fragment_ctor,
    .create_obj_cb = sample_fragment_create_obj,
    .instance_size = sizeof(sample_fragment_t),
};

static lv_obj_t * container = NULL;

void lv_example_fragment_2(void)
{
    lv_obj_t * root = lv_obj_create(lv_screen_active());
    lv_obj_set_size(root, LV_PCT(100), LV_PCT(100));
    lv_obj_set_layout(root, LV_LAYOUT_GRID);
    static const int32_t col_dsc[] = {LV_GRID_FR(1), LV_GRID_FR(1), LV_
↪GRID_TEMPLATE_LAST};
    static const int32_t row_dsc[] = {LV_GRID_FR(1), LV_GRID_CONTENT, LV_
↪GRID_TEMPLATE_LAST};
    lv_obj_set_grid_dsc_array(root, col_dsc, row_dsc);
    container = lv_obj_create(root);
    lv_obj_remove_style_all(container);
    lv_obj_set_grid_cell(container, LV_GRID_ALIGN_STRETCH, 0, 2, LV_GRID_
↪ALIGN_STRETCH, 0, 1);

    lv_obj_t * push_btn = lv_button_create(root);
    lv_obj_t * push_label = lv_label_create(push_btn);
    lv_label_set_text(push_label, "Push");

    lv_obj_t * pop_btn = lv_button_create(root);
    lv_obj_t * pop_label = lv_label_create(pop_btn);
    lv_label_set_text(pop_label, "Pop");
    lv_obj_set_grid_cell(push_btn, LV_GRID_ALIGN_START, 0, 1, LV_GRID_
↪ALIGN_CENTER, 1, 1);
    lv_obj_set_grid_cell(pop_btn, LV_GRID_ALIGN_END, 1, 1, LV_GRID_ALIGN_
↪CENTER, 1, 1);

    lv_fragment_manager_t * manager = lv_fragment_manager_create(NULL);
    /* Clean up the fragment manager before objects in containers got␣
↪deleted */
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_add_event_cb(root, sample_container_delete, LV_EVENT_DELETE,␣
↪manager);

    int depth = 0;
    lv_fragment_t * fragment = lv_fragment_create(&sample_cls, &depth);
    lv_fragment_manager_push(manager, fragment, &container);
    lv_obj_add_event_cb(push_btn, sample_push_click, LV_EVENT_CLICKED,␣
↪manager);
    lv_obj_add_event_cb(pop_btn, sample_pop_click, LV_EVENT_CLICKED,␣
↪manager);
}

static void sample_fragment_ctor(lv_fragment_t * self, void * args)
{
    LV_UNUSED(args);
    ((sample_fragment_t *) self)->depth = *((int *) args);
    ((sample_fragment_t *) self)->counter = 0;
}

static lv_obj_t * sample_fragment_create_obj(lv_fragment_t * self, lv_obj_
↪t * parent)
{
    sample_fragment_t * fragment = (sample_fragment_t *) self;
    lv_obj_t * content = lv_obj_create(parent);
    lv_obj_remove_style_all(content);
    lv_obj_set_style_bg_opa(content, LV_OPA_50, 0);
    lv_obj_set_style_bg_color(content, lv_palette_main(LV_PALETTE_YELLOW),␣
↪0);
    lv_obj_set_size(content, LV_PCT(100), LV_PCT(100));
    lv_obj_set_flex_flow(content, LV_FLEX_FLOW_COLUMN);
    lv_obj_t * depth = lv_label_create(content);
    lv_label_set_text_fmt(depth, "Depth: %d", fragment->depth);
    lv_obj_t * label = lv_label_create(content);
    fragment->label = label;
    lv_label_set_text_fmt(label, "The button has been pressed %d times",␣
↪fragment->counter);

    lv_obj_t * inc_btn = lv_button_create(content);
    lv_obj_t * inc_label = lv_label_create(inc_btn);
    lv_label_set_text(inc_label, "+1");
    lv_obj_add_event_cb(inc_btn, sample_fragment_inc_click, LV_EVENT_
↪CLICKED, fragment);

    return content;
}

static void sample_push_click(lv_event_t * e)
{
    lv_fragment_manager_t * manager = (lv_fragment_manager_t *) lv_event_
↪get_user_data(e);
    size_t stack_size = lv_fragment_manager_get_stack_size(manager);
    lv_fragment_t * fragment = lv_fragment_create(&sample_cls, &stack_
↪size);
    lv_fragment_manager_push(manager, fragment, &container);
}
```

(continues on next page)

(continua dalla pagina precedente)

```
static void sample_pop_click(lv_event_t * e)
{
    lv_fragment_manager_t * manager = (lv_fragment_manager_t *) lv_event_
↪get_user_data(e);
    lv_fragment_manager_pop(manager);
}

static void sample_container_delete(lv_event_t * e)
{
    lv_fragment_manager_t * manager = (lv_fragment_manager_t *) lv_event_
↪get_user_data(e);
    lv_fragment_manager_delete(manager);
}

static void sample_fragment_inc_click(lv_event_t * e)
{
    sample_fragment_t * fragment = (sample_fragment_t *) lv_event_get_user_
↪data(e);
    fragment->counter++;
    lv_label_set_text_fmt(fragment->label, "The button has been pressed %d␣
↪times", fragment->counter);
}

#endif
```

9.3.3 API

lv_fragment.h

lv_fragment_private.h

9.4 Grid navigation

Grid navigation (gridnav for short) is a feature that moves focus among a set of child Widgets via arrow-key presses.

If the child Widgets are arranged into a grid-like layout then the up, down, left and right arrows move focus to the nearest sibling in the respective direction.

It doesn't matter how the children are positioned, as only the current x and y coordinates are considered. This means that gridnav works with manually positioned children, as well as *Flex* and *Grid* layouts.

Gridnav also works if the children are arranged into a single row or column. This is useful, for example, to simplify navigation among items in a *List Widget*.

Gridnav assumes that the Widget to which gridnav is added is part of a *group*. This way, if the Widget with gridnav has focus, the arrow key presses are automatically forwarded to the Widget so that gridnav can process the arrow keys.

To move the focus to the next widget of the group use `LV_KEY_NEXT` or `LV_KEY_PREV`. Optionally you can also use `lv_group_focus_next()` or `lv_group_focus_prev()` or the TAB key on keyboard as usual.

If the container is scrollable and the focused child is out of the view, gridnav will automatically scroll the child into view.

9.4.1 Utilizzo

To add gridnav behavior to any Widget (e.g. one serving as a container for child Widgets that the end user will navigate among using arrow keys) use `lv_gridnav_add(container, flags)`.

The `flags` argument controls the navigation behavior:

- *LV_GRIDNAV_CTRL_NONE*: Default settings
- *LV_GRIDNAV_CTRL_ROLLOVER*: If there is no next/previous Widget in a direction, the focus goes to the Widget in the next/previous row (on left/right keys) or first/last row (on up/down keys)
- *LV_GRIDNAV_CTRL_SCROLL_FIRST*: If an arrow is pressed and the focused Widget can be scrolled in that direction then it will be scrolled instead of going to the next/previous Widget. If there is no more room for scrolling the next/previous Widget will receive focus normally.
- *LV_GRIDNAV_CTRL_HORIZONTAL_MOVE_ONLY*: Only use the left/right keys for grid navigation. Up/down key events will be sent to the Widget that has focus.
- *LV_GRIDNAV_CTRL_VERTICAL_MOVE_ONLY*: Only use the up/down keys for grid navigation. Left/right key events will be sent to the Widget that has focus.

While the above behaviors can be combined by bit-wise OR-ing the above values together, *LV_GRIDNAV_CTRL_HORIZONTAL_MOVE_ONLY* and *LV_GRIDNAV_CTRL_VERTICAL_MOVE_ONLY* should not be used together.

lv_gridnav_remove(container) Removes gridnav behavior from a Widget.

9.4.2 Focusable Widgets

A Widget needs to be clickable or click focusable (*LV_OBJ_FLAG_CLICKABLE* or *LV_OBJ_FLAG_CLICK_FOCUSABLE*) and not hidden (*LV_OBJ_FLAG_HIDDEN*) to receive focus via gridnav.

9.4.3 Esempi

Basic grid navigation

```
#include "../../lv_examples.h"
#if LV_USE_GRIDNAV && LV_USE_FLEX && LV_BUILD_EXAMPLES

/**
 * Demonstrate a a basic grid navigation
 */
void lv_example_gridnav_1(void)
{
    /*It's assumed that the default group is set and
     *there is a keyboard indev*/

    lv_obj_t * cont1 = lv_obj_create(lv_screen_active());
    lv_gridnav_add(cont1, LV_GRIDNAV_CTRL_NONE);

    /*Use flex here, but works with grid or manually placed objects as␣
↪well*/
    lv_obj_set_flex_flow(cont1, LV_FLEX_FLOW_ROW_WRAP);
    lv_obj_set_style_bg_color(cont1, lv_palette_lighten(LV_PALETTE_BLUE,␣
↪5), LV_STATE_FOCUSED);
    lv_obj_set_size(cont1, lv_pct(50), lv_pct(100));

    /*Only the container needs to be in a group*/
    lv_group_add_obj(lv_group_get_default(), cont1);

    lv_obj_t * label = lv_label_create(cont1);
    lv_label_set_text_fmt(label, "No rollover");

    uint32_t i;
    for(i = 0; i < 10; i++) {
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_obj_t * obj = lv_button_create(cont1);
        lv_obj_set_size(obj, 70, LV_SIZE_CONTENT);
        lv_obj_add_flag(obj, LV_OBJ_FLAG_CHECKABLE);
        lv_group_remove_obj(obj);   /*Not needed, we use the gridnav␣
↪instead*/

        label = lv_label_create(obj);
        lv_label_set_text_fmt(label, "%" LV_PRIu32"", i);
        lv_obj_center(label);
    }

    /* Create a second container with rollover grid nav mode.*/

    lv_obj_t * cont2 = lv_obj_create(lv_screen_active());
    lv_gridnav_add(cont2, LV_GRIDNAV_CTRL_ROLLOVER);
    lv_obj_set_style_bg_color(cont2, lv_palette_lighten(LV_PALETTE_BLUE,␣
↪5), LV_STATE_FOCUSED);
    lv_obj_set_size(cont2, lv_pct(50), lv_pct(100));
    lv_obj_align(cont2, LV_ALIGN_RIGHT_MID, 0, 0);

    label = lv_label_create(cont2);
    lv_obj_set_width(label, lv_pct(100));
    lv_label_set_text_fmt(label, "Rollover\nUse tab to focus the other␣
↪container");

    /*Only the container needs to be in a group*/
    lv_group_add_obj(lv_group_get_default(), cont2);

    /*Add and place some children manually*/
    lv_obj_t * ta = lv_textarea_create(cont2);
    lv_obj_set_size(ta, lv_pct(100), 80);
    lv_obj_set_pos(ta, 0, 80);
    lv_group_remove_obj(ta);   /*Not needed, we use the gridnav instead*/

    lv_obj_t * cb = lv_checkbox_create(cont2);
    lv_obj_set_pos(cb, 0, 170);
    lv_group_remove_obj(cb);   /*Not needed, we use the gridnav instead*/

    lv_obj_t * sw1 = lv_switch_create(cont2);
    lv_obj_set_pos(sw1, 0, 200);
    lv_group_remove_obj(sw1);   /*Not needed, we use the gridnav instead*/

    lv_obj_t * sw2 = lv_switch_create(cont2);
    lv_obj_set_pos(sw2, lv_pct(50), 200);
    lv_group_remove_obj(sw2);   /*Not needed, we use the gridnav instead*/
}

#endif
```

Grid navigation on a list

```
#include "../../lv_examples.h"
#if LV_USE_GRIDNAV && LV_USE_LIST && LV_BUILD_EXAMPLES

/**
 * Grid navigation on a list
```

(continues on next page)

(continua dalla pagina precedente)

```
 */
void lv_example_gridnav_2(void)
{
    /*It's assumed that the default group is set and
     *there is a keyboard indev*/

    lv_obj_t * list1 = lv_list_create(lv_screen_active());
    lv_gridnav_add(list1, LV_GRIDNAV_CTRL_NONE);
    lv_obj_set_size(list1, lv_pct(45), lv_pct(80));
    lv_obj_align(list1, LV_ALIGN_LEFT_MID, 5, 0);
    lv_obj_set_style_bg_color(list1, lv_palette_lighten(LV_PALETTE_BLUE,␣
↪5), LV_STATE_FOCUSED);
    lv_group_add_obj(lv_group_get_default(), list1);

    char buf[32];
    uint32_t i;
    for(i = 0; i < 15; i++) {
        lv_snprintf(buf, sizeof(buf), "File %d", i + 1);
        lv_obj_t * item = lv_list_add_button(list1, LV_SYMBOL_FILE, buf);
        lv_obj_set_style_bg_opa(item, 0, 0);
        lv_group_remove_obj(item);   /*Not needed, we use the gridnav␣
↪instead*/
    }

    lv_obj_t * list2 = lv_list_create(lv_screen_active());
    lv_gridnav_add(list2, LV_GRIDNAV_CTRL_ROLLOVER);
    lv_obj_set_size(list2, lv_pct(45), lv_pct(80));
    lv_obj_align(list2, LV_ALIGN_RIGHT_MID, -5, 0);
    lv_obj_set_style_bg_color(list2, lv_palette_lighten(LV_PALETTE_BLUE,␣
↪5), LV_STATE_FOCUSED);
    lv_group_add_obj(lv_group_get_default(), list2);

    for(i = 0; i < 15; i++) {
        lv_snprintf(buf, sizeof(buf), "Folder %d", i + 1);
        lv_obj_t * item = lv_list_add_button(list2, LV_SYMBOL_DIRECTORY,␣
↪buf);
        lv_obj_set_style_bg_opa(item, 0, 0);
        lv_group_remove_obj(item);
    }
}

#endif
```

Nested grid navigations

```
#include "../../lv_examples.h"
#if LV_USE_GRIDNAV && LV_USE_FLEX && LV_BUILD_EXAMPLES

static void cont_sub_event_cb(lv_event_t * e)
{
    uint32_t k = lv_event_get_key(e);
    lv_obj_t * obj = lv_event_get_target_obj(e);
    if(k == LV_KEY_ENTER) {
        lv_group_focus_obj(obj);
    }
    else if(k == LV_KEY_ESC) {
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_group_focus_next(lv_obj_get_group(obj));
    }

}

/**
 * Nested grid navigations
 */
void lv_example_gridnav_3(void)
{
    /*It's assumed that the default group is set and
     *there is a keyboard indev*/

    lv_obj_t * cont_main = lv_obj_create(lv_screen_active());
    lv_gridnav_add(cont_main, (lv_gridnav_ctrl_t)(LV_GRIDNAV_CTRL_ROLLOVER␣
↪| LV_GRIDNAV_CTRL_SCROLL_FIRST));

    /*Only the container needs to be in a group*/
    lv_group_add_obj(lv_group_get_default(), cont_main);

    /*Use flex here, but works with grid or manually placed objects as␣
↪well*/
    lv_obj_set_flex_flow(cont_main, LV_FLEX_FLOW_ROW_WRAP);
    lv_obj_set_style_bg_color(cont_main, lv_palette_lighten(LV_PALETTE_
↪BLUE, 5), LV_STATE_FOCUSED);
    lv_obj_set_size(cont_main, lv_pct(80), LV_SIZE_CONTENT);

    lv_obj_t * btn;
    lv_obj_t * label;

    btn = lv_button_create(cont_main);
    lv_group_remove_obj(btn);
    label = lv_label_create(btn);
    lv_label_set_text(label, "Button 1");

    btn = lv_button_create(cont_main);
    lv_group_remove_obj(btn);
    label = lv_label_create(btn);
    lv_label_set_text(label, "Button 2");

    /*Create another container with long text to show how LV_GRIDNAV_CTRL_
↪SCROLL_FIRST works*/
    lv_obj_t * cont_sub1 = lv_obj_create(cont_main);
    lv_obj_set_size(cont_sub1, lv_pct(100), 100);

    label = lv_label_create(cont_sub1);
    lv_obj_set_style_bg_color(cont_sub1, lv_palette_lighten(LV_PALETTE_RED,
↪ 5), LV_STATE_FOCUSED);
    lv_obj_set_width(label, lv_pct(100));
    lv_label_set_text(label,
                      "I'm a very long text which is makes my container␣
↪scrollable. "
                      "As LV_GRIDNAV_FLAG_SCROLL_FIRST is enabled arrow␣
↪will scroll me first "
                      "and a new objects will be focused only when an edge␣
↪is reached with the scrolling.\n\n"
```

(continues on next page)

(continua dalla pagina precedente)

```
                        "This is only some placeholder text to be sure the␣
↪parent will be scrollable. \n\n"
                        "Hello world!\n"
                        "Hello world!\n"
                        "Hello world!\n"
                        "Hello world!\n"
                        "Hello world!\n"
                        "Hello world!");

    /*Create a third container that can be focused with ENTER and contains␣
↪another grid nav*/
    lv_obj_t * cont_sub2 = lv_obj_create(cont_main);
    lv_gridnav_add(cont_sub2, LV_GRIDNAV_CTRL_ROLLOVER);
    /*Only the container needs to be in a group*/
    lv_group_add_obj(lv_group_get_default(), cont_sub2);

    lv_obj_add_event_cb(cont_sub2, cont_sub_event_cb, LV_EVENT_KEY, NULL);

    /*Use flex here, but works with grid or manually placed objects as␣
↪well*/
    lv_obj_set_flex_flow(cont_sub2, LV_FLEX_FLOW_ROW_WRAP);
    lv_obj_set_style_bg_color(cont_sub2, lv_palette_lighten(LV_PALETTE_RED,
↪ 5), LV_STATE_FOCUSED);
    lv_obj_set_size(cont_sub2, lv_pct(100), LV_SIZE_CONTENT);

    label = lv_label_create(cont_sub2);
    lv_label_set_text(label, "Use ENTER/ESC to focus/defocus this container
↪");
    lv_obj_set_width(label, lv_pct(100));

    btn = lv_button_create(cont_sub2);
    lv_group_remove_obj(btn);
    label = lv_label_create(btn);
    lv_label_set_text(label, "Button 3");

    btn = lv_button_create(cont_sub2);
    lv_group_remove_obj(btn);
    label = lv_label_create(btn);
    lv_label_set_text(label, "Button 4");

}

#endif
```

Simple navigation on a list widget

```
#include "../../lv_examples.h"
#if LV_USE_GRIDNAV && LV_USE_FLEX && LV_BUILD_EXAMPLES

static void event_handler(lv_event_t * e)
{
    lv_obj_t * obj = lv_event_get_target_obj(e);
    lv_obj_t * list = lv_obj_get_parent(obj);
    LV_UNUSED(list); /*If logging is disabled*/
    LV_LOG_USER("Clicked: %s", lv_list_get_button_text(list, obj));
}
```

(continues on next page)

(continua dalla pagina precedente)

```
/**
 * Simple navigation on a list widget
 */
void lv_example_gridnav_4(void)
{
    /*It's assumed that the default group is set and
     *there is a keyboard indev*/

    lv_obj_t * list = lv_list_create(lv_screen_active());
    lv_gridnav_add(list, LV_GRIDNAV_CTRL_ROLLOVER);
    lv_obj_align(list, LV_ALIGN_LEFT_MID, 10, 0);
    lv_group_add_obj(lv_group_get_default(), list);

    uint32_t i;
    for(i = 0; i < 20; i++) {
        char buf[32];

        /*Add some separators too, they are not focusable by gridnav*/
        if((i % 5) == 0) {
            lv_snprintf(buf, sizeof(buf), "Section %d", i / 5 + 1);
            lv_list_add_text(list, buf);
        }

        lv_snprintf(buf, sizeof(buf), "File %d", i + 1);
        lv_obj_t * item = lv_list_add_button(list, LV_SYMBOL_FILE, buf);
        lv_obj_add_event_cb(item, event_handler, LV_EVENT_CLICKED, NULL);
        lv_group_remove_obj(item);  /*The default group adds it␣
↪automatically*/
    }

    lv_obj_t * btn = lv_button_create(lv_screen_active());
    lv_obj_align(btn, LV_ALIGN_RIGHT_MID, -10, 0);
    lv_obj_t * label = lv_label_create(btn);
    lv_label_set_text(label, "Button");
}

#endif
```

Grid navigation for only one axis

```
#include "../../lv_examples.h"
#if LV_USE_GRIDNAV && LV_USE_FLEX && LV_BUILD_EXAMPLES

static const char * opts[] = {"0\n1\n2\n3\n4\n5", "0\n1\n2\n3\n4\n5\n6\n7\
↪n8\n9", "s\nm\nh"};
static const int32_t opts_counts[] = {6, 10, 3};

static lv_obj_t * sliders[3];
static lv_obj_t * rollers[3];

static void slider_key_cb(lv_event_t * e)
{
    uint8_t i = (uint32_t)(uintptr_t)lv_event_get_user_data(e);
    lv_roller_set_selected(rollers[i], lv_slider_get_value(sliders[i]), LV_
↪ANIM_ON);
```

(continues on next page)

(continua dalla pagina precedente)

```
}
static void roller_key_cb(lv_event_t * e)
{
    uint8_t i = (uint32_t)(uintptr_t)lv_event_get_user_data(e);
    lv_slider_set_value(sliders[i], lv_roller_get_selected(rollers[i]), LV_
↪ANIM_ON);
}

/**
 * Grid navigation for only one axis
 */
void lv_example_gridnav_5(void)
{
    /*It's assumed that the default group is set and
     *there is a keyboard indev*/

    lv_group_t * group = lv_group_get_default();
    lv_obj_t * cont;

    cont = lv_obj_create(lv_screen_active());
    lv_obj_set_flex_flow(cont, LV_FLEX_FLOW_COLUMN);
    lv_obj_set_flex_align(cont, LV_FLEX_ALIGN_CENTER, LV_FLEX_ALIGN_CENTER,
↪ LV_FLEX_ALIGN_CENTER);
    lv_obj_set_size(cont, lv_pct(100), lv_pct(50));
    lv_obj_align(cont, LV_ALIGN_TOP_MID, 0, 0);
    /* only up/down keys will be used for grid navigation in this␣
↪container. */
    /* left/right will be sent to the sliders */
    lv_gridnav_add(cont, LV_GRIDNAV_CTRL_VERTICAL_MOVE_ONLY);
    lv_group_add_obj(group, cont);
    for(uint32_t i = 0; i < 3; i++) {
        lv_obj_t * slider = lv_slider_create(cont);
        lv_slider_set_range(slider, 0, opts_counts[i] - 1);
        lv_group_remove_obj(slider);
        lv_obj_set_width(slider, lv_pct(85));
        sliders[i] = slider;
        lv_obj_add_event_cb(slider, slider_key_cb, LV_EVENT_KEY, (void␣
↪*)(uintptr_t)i);
    }

    cont = lv_obj_create(lv_screen_active());
    lv_obj_set_flex_flow(cont, LV_FLEX_FLOW_ROW);
    lv_obj_set_flex_align(cont, LV_FLEX_ALIGN_CENTER, LV_FLEX_ALIGN_CENTER,
↪ LV_FLEX_ALIGN_CENTER);
    lv_obj_set_size(cont, lv_pct(100), lv_pct(50));
    lv_obj_align(cont, LV_ALIGN_BOTTOM_MID, 0, 0);
    /* only left/right keys will be used for grid navigation in this␣
↪container. */
    /* up/down will be sent to the rollers */
    lv_gridnav_add(cont, LV_GRIDNAV_CTRL_HORIZONTAL_MOVE_ONLY);
    lv_group_add_obj(group, cont);
    for(uint32_t i = 0; i < 3; i++) {
        lv_obj_t * roller = lv_roller_create(cont);
        lv_roller_set_options(roller, opts[i], LV_ROLLER_MODE_INFINITE);
        lv_obj_set_size(roller, lv_pct(30), lv_pct(100));
        lv_group_remove_obj(roller);
```

(continues on next page)

(continua dalla pagina precedente)

```
        rollers[i] = roller;
        lv_obj_add_event_cb(roller, roller_key_cb, LV_EVENT_KEY, (void␣
↪*)(uintptr_t)i);
    }
}

#endif
```

9.4.4 API

lv_gridnav.h

9.5 Pinyin IME

Pinyin IME provides an API to provide Chinese Pinyin input method (Chinese input) for a Keyboard Widget, which supports both 26-key and 9-key input modes. You can think of `lv_ime_pinyin` as a Pinyin input method plug-in for the Keyboard Widget.

Normally, an environment where *Keyboard (lv_keyboard)* can run can also run `lv_ime_pinyin`. There are two main influencing factors: the size of the font file and the size of the dictionary.

9.5.1 Utilizzo

Enable *`LV_USE_IME_PINYIN`* in `lv_conf.h`.

First use *`lv_ime_pinyin_create`*`(`*`lv_screen_active`*`())` to create a Pinyin input-method plug-in, then use *`lv_ime_pinyin_set_keyboard`*`(pinyin_ime, kb)` to add the Keyboard Widget you created to the Pinyin input method plug-in. You can use *`lv_ime_pinyin_set_dict`*`(pinyin_ime, your_dict)` to use a custom dictionary. If you don't want to use the built-in dictionary, you can disable `LV_IME_PINYIN_USE_DEFAULT_DICT` in `lv_conf.h`, which can save a lot of memory space.

The built-in thesaurus is customized based on the **LV_FONT_SOURCE_HAN_SANS_SC_16_CJK** font library, which currently has more than 1,000 of the most common CJK radicals, so it is recommended to use a custom font and thesaurus.

In the process of using the Pinyin input method plug-in, you can change the Keyboard and dictionary at any time.

9.5.2 Custom Dictionary

If you don't want to use the built-in Pinyin dictionary, or if you feel that the built-in phonetic dictionary consumes too much memory, you can use a custom dictionary.

Customizing the dictionary is very simple.

First, set `LV_IME_PINYIN_USE_DEFAULT_DICT` to `0` in `lv_conf.h`

Then, write a dictionary in the following format.

Dictionary format

The arrangement order of each pinyin syllable is very important. If you need to customize your own thesaurus according to the Hanyu Pinyin syllable table, you can read here to learn about the Hanyu Pinyin syllables and the syllable table.

Then, write your own dictionary according to the following format:

```
lv_100ask_pinyin_dict_t your_pinyin_dict[] = {
            { "a", "□□□□" },
            { "ai", "□□□□□□□□□□□□□□" },
```

(continues on next page)

(continua dalla pagina precedente)

```
            { "an", "□□□□□□□□□□□□□□□□□□□□□□" },
            { "ang", "□□□□" },
            { "ao", "□□□□□□□□□□" },
            { "ba", "□□□□□□□□□□□□□□□□□□□□" },
            { "bai", "□□□□□□□□□□" },
            /* ...... */
            { "zuo", "□□□□□□□□□□□"},
            {NULL, NULL}
```

The last item must be `{null, null}`, or it will not work properly.

Applying a new dictionary

After writing a dictionary according to the above dictionary format, you only need to call this function to set up and use your dictionary:

```
lv_obj_t * pinyin_ime = lv_100ask_pinyin_ime_create(lv_screen_active());
lv_100ask_pinyin_ime_set_dict(pinyin_ime, your_pinyin_dict);
```

9.5.3 Modes

lv_ime_pinyin has the following modes:

- *LV_IME_PINYIN_MODE_K26*: Pinyin 26-key input mode
- *LV_IME_PINYIN_MODE_K9*: Pinyin 9-key input mode
- *LV_IME_PINYIN_MODE_K9_NUMBER*: Numeric keypad mode

The Keyboard's TEXT-mode layout contains buttons to change mode.

To set the mode manually, use *lv_ime_pinyin_set_mode*`(pinyin_ime, mode)`. The default mode is *LV_IME_PINYIN_MODE_K26*.

9.5.4 Esempio

Pinyin IME 26 key input

```
#include "../../lv_examples.h"
#if LV_USE_LABEL && LV_USE_TEXTAREA && LV_FONT_SOURCE_HAN_SANS_SC_16_CJK &&
↪ LV_USE_IME_PINYIN && LV_BUILD_EXAMPLES

static void ta_event_cb(lv_event_t * e)
{
    lv_event_code_t code = lv_event_get_code(e);
    lv_obj_t * ta = lv_event_get_target_obj(e);
    lv_obj_t * kb = (lv_obj_t *) lv_event_get_user_data(e);

    if(code == LV_EVENT_FOCUSED) {
        if(lv_indev_get_type(lv_indev_active()) != LV_INDEV_TYPE_KEYPAD) {
            lv_keyboard_set_textarea(kb, ta);
            lv_obj_remove_flag(kb, LV_OBJ_FLAG_HIDDEN);
        }
    }
    else if(code == LV_EVENT_CANCEL) {
        lv_obj_add_flag(kb, LV_OBJ_FLAG_HIDDEN);
        lv_obj_remove_state(ta, LV_STATE_FOCUSED);
        lv_indev_reset(NULL, ta);   /*To forget the last clicked object to␣
↪make it focusable again*/
```

(continues on next page)

(continua dalla pagina precedente)

```
    }
}

void lv_example_ime_pinyin_1(void)
{
    lv_obj_t * pinyin_ime = lv_ime_pinyin_create(lv_screen_active());
    lv_obj_set_style_text_font(pinyin_ime, &lv_font_source_han_sans_sc_16_
↪cjk, 0);
    //lv_ime_pinyin_set_dict(pinyin_ime, your_dict); // Use a custom␣
↪dictionary. If it is not set, the built-in dictionary will be used.

    /* ta1 */
    lv_obj_t * ta1 = lv_textarea_create(lv_screen_active());
    lv_textarea_set_one_line(ta1, true);
    lv_obj_set_style_text_font(ta1, &lv_font_source_han_sans_sc_16_cjk, 0);
    lv_obj_align(ta1, LV_ALIGN_TOP_LEFT, 0, 0);

    /*Create a keyboard and add it to ime_pinyin*/
    lv_obj_t * kb = lv_keyboard_create(lv_screen_active());
    lv_ime_pinyin_set_keyboard(pinyin_ime, kb);
    lv_keyboard_set_textarea(kb, ta1);

    lv_obj_add_event_cb(ta1, ta_event_cb, LV_EVENT_ALL, kb);

    /*Get the cand_panel, and adjust its size and position*/
    lv_obj_t * cand_panel = lv_ime_pinyin_get_cand_panel(pinyin_ime);
    lv_obj_set_size(cand_panel, LV_PCT(100), LV_PCT(10));
    lv_obj_align_to(cand_panel, kb, LV_ALIGN_OUT_TOP_MID, 0, 0);

    /*Try using ime_pinyin to output the Chinese below in the ta1 above*/
    lv_obj_t * cz_label = lv_label_create(lv_screen_active());
    lv_label_set_text(cz_label,
                      "□□□□□□Embedded System□□\
↪n□□□□□□
```

(continua dalla pagina precedente)

```
            lv_keyboard_set_textarea(kb, ta);
            lv_obj_remove_flag(kb, LV_OBJ_FLAG_HIDDEN);
        }
    }
    else if(code == LV_EVENT_READY) {
        lv_obj_add_flag(kb, LV_OBJ_FLAG_HIDDEN);
        lv_obj_remove_state(ta, LV_STATE_FOCUSED);
        lv_indev_reset(NULL, ta);   /*To forget the last clicked object to␣
↪make it focusable again*/
    }
}

void lv_example_ime_pinyin_2(void)
{
    lv_obj_t * pinyin_ime = lv_ime_pinyin_create(lv_screen_active());
    lv_obj_set_style_text_font(pinyin_ime, &lv_font_source_han_sans_sc_16_
↪cjk, 0);
    //lv_ime_pinyin_set_dict(pinyin_ime, your_dict); // Use a custom␣
↪dictionary. If it is not set, the built-in dictionary will be used.

    /* ta1 */
    lv_obj_t * ta1 = lv_textarea_create(lv_screen_active());
    lv_textarea_set_one_line(ta1, true);
    lv_obj_set_style_text_font(ta1, &lv_font_source_han_sans_sc_16_cjk, 0);
    lv_obj_align(ta1, LV_ALIGN_TOP_LEFT, 0, 0);

    /*Create a keyboard and add it to ime_pinyin*/
    lv_obj_t * kb = lv_keyboard_create(lv_screen_active());
    lv_keyboard_set_textarea(kb, ta1);

    lv_ime_pinyin_set_keyboard(pinyin_ime, kb);
    lv_ime_pinyin_set_mode(pinyin_ime,
                           LV_IME_PINYIN_MODE_K9);  // Set to 9-key input␣
↪mode. Default: 26-key input(k26) mode.
    lv_obj_add_event_cb(ta1, ta_event_cb, LV_EVENT_ALL, kb);

    /*Get the cand_panel, and adjust its size and position*/
    lv_obj_t * cand_panel = lv_ime_pinyin_get_cand_panel(pinyin_ime);
    lv_obj_set_size(cand_panel, LV_PCT(100), LV_PCT(10));
    lv_obj_align_to(cand_panel, kb, LV_ALIGN_OUT_TOP_MID, 0, 0);

    /*Try using ime_pinyin to output the Chinese below in the ta1 above*/
    lv_obj_t * cz_label = lv_label_create(lv_screen_active());
    lv_label_set_text(cz_label,
                      "□□□□□□Embedded System□□\
↪n□□□□□□
```

9.5.5 API

lv_ime_pinyin.h

lv_ime_pinyin_private.h

9.6 Image font

Draw image in **label** or **span** obj with `lv_imgfont`. This is often used to display Unicode emoji icons in text.

Supported image formats: determined by enabled LVGL *image decoders*.

9.6.1 Utilizzo

Enable `LV_USE_IMGFONT` in `lv_conf.h`.

To create a new *imgfont*:

```
static lv_font_t  * imgfont;
...
imgfont = lv_imgfont_create(height, path_cb, user_data);
```

- `height` Font size.
- `path_cb` A function to get the image path of a character. Pass `NULL` if no image should be shown, but the character itself.
- `user_data` Pointer to user data.

To use the *imgfont* in a label, reference it: `lv_obj_set_style_text_font(label, imgfont, LV_PART_MAIN)`

To destroy the *imgfont* that is no longer used, use `lv_imgfont_destroy(imgfont)`.

9.6.2 Esempio

Use emojis in a text.

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

#if LV_BUILD_EXAMPLES
#if LV_USE_IMGFONT

static const void * get_imgfont_path(const lv_font_t * font, uint32_t␣
↪unicode, uint32_t unicode_next,
                                     int32_t * offset_y, void * user_data)
{
    LV_UNUSED(font);
    LV_UNUSED(unicode_next);
    LV_UNUSED(offset_y);
    LV_UNUSED(user_data);

    LV_IMAGE_DECLARE(emoji_F617);

    if(unicode < 0xF000) return NULL;

    if(unicode == 0xF617) {
        return &emoji_F617;
    }
    else if(unicode == 0xF600) {
#if LV_USE_FFMPEG
```

(continues on next page)

(continua dalla pagina precedente)

```
        return "lvgl/examples/assets/emoji/F600.png";
#else
        return "A:lvgl/examples/assets/emoji/F600.png";
#endif
    }

    return NULL;
}

/**
 * draw img in label or span obj
 */
void lv_example_imgfont_1(void)
{
    lv_font_t * imgfont = lv_imgfont_create(80, get_imgfont_path, NULL);
    if(imgfont == NULL) {
        LV_LOG_ERROR("imgfont init error");
        return;
    }

    imgfont->fallback = LV_FONT_DEFAULT;

    lv_obj_t * label1 = lv_label_create(lv_screen_active());
    lv_label_set_text(label1, "12\uF600\uF617AB");
    lv_obj_set_style_text_font(label1, imgfont, LV_PART_MAIN);
    lv_obj_center(label1);
}
#else

void lv_example_imgfont_1(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "imgfont is not installed");
    lv_obj_center(label);
}

#endif
#endif
```

9.6.3 API

lv_imgfont.h

9.7 Monkey

The Monkey module provides LVGL applications with a simple monkey test. Monkey Testing is a technique where the user tests the application or system by providing random inputs and checking the behavior or seeing whether the application or system will crash. This module provides this service as simulated random input to stress test an LVGL application.

9.7.1 Utilizzo

First, enable *LV_USE_MONKEY* in `lv_conf.h`.

Next, declare a variable (it can be local) of type `lv_monkey_config_t` to define the configuration structure, initialize it using *`lv_monkey_config_init`*`(cfg)` then set its `type` member to the desired type of *input*

device, and set the `min` and `max` values for its `period_range` and `input_range` members to set the time ranges (in milliseconds) and input ranges the Monkey module will use to generate random input at random times.

Next, call `lv_monkey_create(cfg)` to create the Monkey. It returns a pointer to the `lv_monkey_t` created.

Finally call `lv_monkey_set_enable(monkey, true)` to enable Monkey.

If you want to pause the monkey, call `lv_monkey_set_enable(monkey, false)`. To delete it, call `lv_monkey_delete(monkey)`.

Note that `input_range` has different meanings depending on the `type` input device:

- `LV_INDEV_TYPE_POINTER`: No effect, click randomly within the pixels of the screen resolution.
- `LV_INDEV_TYPE_ENCODER`: The minimum and maximum values of `enc_diff`.
- `LV_INDEV_TYPE_BUTTON`: The minimum and maximum values of `btn_id`. Use `lv_monkey_get_indev()` to get the input device, and use `lv_indev_set_button_points()` to map the key ID to the coordinates.
- `LV_INDEV_TYPE_KEYPAD`: No effect, Send random *Keys*.

9.7.2 Esempio

Touchpad monkey example

```
#include "../../lv_examples.h"
#if LV_USE_MONKEY && LV_BUILD_EXAMPLES

void lv_example_monkey_1(void)
{
    /*Create pointer monkey test*/
    lv_monkey_config_t config;
    lv_monkey_config_init(&config);
    config.type = LV_INDEV_TYPE_POINTER;
    config.period_range.min = 10;
    config.period_range.max = 100;
    lv_monkey_t * monkey = lv_monkey_create(&config);

    /*Start monkey test*/
    lv_monkey_set_enable(monkey, true);
}

#endif
```

Encoder monkey example

```
#include "../../lv_examples.h"
#if LV_USE_MONKEY && LV_BUILD_EXAMPLES

void lv_example_monkey_2(void)
{
    /*Create encoder monkey test*/
    lv_monkey_config_t config;
    lv_monkey_config_init(&config);
    config.type = LV_INDEV_TYPE_ENCODER;
    config.period_range.min = 50;
    config.period_range.max = 500;
    config.input_range.min = -5;
    config.input_range.max = 5;
    lv_monkey_t * monkey = lv_monkey_create(&config);
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Set the default group*/
    lv_group_t * group = lv_group_create();
    lv_indev_set_group(lv_monkey_get_indev(monkey), group);
    lv_group_set_default(group);

    /*Start monkey test*/
    lv_monkey_set_enable(monkey, true);
}

#endif
```

Button monkey example

```
#include "../../lv_examples.h"
#if LV_USE_MONKEY && LV_BUILD_EXAMPLES

void lv_example_monkey_3(void)
{
    static lv_point_t btn_points[3];
    int32_t hor_res = LV_HOR_RES;

    /*Create button monkey test*/
    lv_monkey_config_t config;
    lv_monkey_config_init(&config);
    config.type = LV_INDEV_TYPE_BUTTON;
    config.period_range.min = 50;
    config.period_range.max = 500;
    config.input_range.min = 0;
    config.input_range.max = sizeof(btn_points) / sizeof(lv_point_t) - 1;
    lv_monkey_t * monkey = lv_monkey_create(&config);

    /*Set the coordinates bound to the button*/
    btn_points[0].x = hor_res / 4;
    btn_points[0].y = 10;
    btn_points[1].x = hor_res / 2;
    btn_points[1].y = 10;
    btn_points[2].x = hor_res * 3 / 4;
    btn_points[2].y = 10;

    lv_indev_set_button_points(lv_monkey_get_indev(monkey), btn_points);

    /*Start monkey test*/
    lv_monkey_set_enable(monkey, true);
}

#endif
```

9.7.3 API

lv_monkey.h

9.8 Widget ID

Widgets can optionally have identifiers added to their functionality if needed for the application. Exactly how that happens is designed to be flexible, and can morph with the needs of the application. It can even be a timestamp or other data current at the time the Widget was created.

9.8.1 Utilizzo

Enable Widget ID functionality by setting *LV_USE_OBJ_ID* to `1` in `lv_conf.h`.

Once enabled, several things change:

- each Widget will now have a `void *` field called `id`;
- these two API functions become available:
  - *lv_obj_get_id*`(widget)`,
  - *lv_obj_find_by_id*`(widget, id)`;
- several more Widget-ID-related API functions become available if *LV_USE_OBJ_ID_BUILTIN* is non-zero (more on this below);
- two additional configuration macros both *LV_OBJ_ID_AUTO_ASSIGN* and *LV_USE_OBJ_ID_BUILTIN* now have meaning.

LV_OBJ_ID_AUTO_ASSIGN

This macro in `lv_conf.h` defaults to whatever value *LV_USE_OBJ_ID* equates to. You can change this if you wish. Either way, if it equates to a non-zero value, it causes two things to happen:

- *lv_obj_assign_id*`(class_p, widget)` will be called at the end of each Widget's creation, and
- *lv_obj_free_id*`(widget)` will be called at the end of the sequence when each Widget is deleted.

Because of this timing, custom versions of these functions can be used according to the below, and they can even be used like "event hooks" to implement a trace operation that occurs when each Widget is created and deleted.

lv_obj_assign_id(class_p, widget)

This function (whether provided by LVGL or by you --- more on this below) is responsible for assigning a value to the Widget's `id` field, and possibly do other things, depending on the implementation.

lv_obj_free_id(widget)

This function (whether provided by LVGL or by you --- more on this below) is responsible for doing the clean-up of any resources allocated by *lv_obj_assign_id()*

LV_USE_OBJ_ID_BUILTIN

This macro in `lv_conf.h` equates to `1` by default. You can change this if you wish. When it equates to a non-zero value the following function implementations are provided by LVGL:

- *lv_obj_assign_id*`(class_p, widget)`
- *lv_obj_free_id*`(widget)`
- *lv_obj_set_id*`(widget, id)`
- *lv_obj_stringify_id*`(widget, buf, len)`
- *lv_obj_id_compare*`(id1, id2)`

These supply the default implementation for Widget IDs, namely that for each Widget created, *lv_obj_stringify_id*`(widget, buf, len)` will produce a unique string for it. Example: if the following 6 Widgets are created in this sequence:

- Screen
- Etichetta
- Pulsante
- Etichetta
- Etichetta
- Immagine

the strings produced by `lv_obj_stringify_id(widget, buf, len)` would be

- obj1
- label1
- btn1
- label2
- label3
- image1

respectively.

Using a custom ID generator

If you wish, you can provide custom implementations for several Widget-ID related functions. You do this by first setting `LV_USE_OBJ_ID_BUILTIN` to `0` in `lv_conf.h`.

You will then need to provide implementations for the following functions (and link them with LVGL):

```
const char * lv_obj_stringify_id(lv_obj_t * widget, char * buf, uint32_t␣
↪len);
int          lv_obj_id_compare(const void * id1, const void * id2);
```

If `LV_OBJ_ID_AUTO_ASSIGN` equates to a non-zero value (or if you otherwise simply need to use them), you will also need to provide implementations for:

```
void         lv_obj_assign_id(const lv_obj_class_t * class_p, lv_obj_t *␣
↪widget);
void         lv_obj_free_id(lv_obj_t * widget);
```

If `LV_BUILD_TEST` equates to a non-zero value and you are including LVGL test code in your compile (or if you otherwise simply need to use them), you will also need to provide an implementation for:

```
void         lv_obj_set_id(lv_obj_t * widget, void * id);
```

Examples of implementations of these functions exist in `lv_obj_id_builtin.c`, but you are free to use a different design if needed.

`lv_obj_stringify_id()` converts the passed `widget` to a string representation (typically incorporating the `id` field) and writes it into the buffer provided in its `buf` argument.

`lv_obj_id_compare()` compares 2 `void * id` values and returns `0` when they are considered equal, and non-zero otherwise.

If `LV_OBJ_ID_AUTO_ASSIGN` equates to a non-zero value, `lv_obj_assign_id()` is called when a Widget is created. It is responsible for assigning a value to the Widget's `id` field. A pointer to the Widget's final class is passed in its `class_p` argument in case it is needed for determining the value for the `id` field, or for other possible needs related to your design for Widget IDs. Note that this pointer may be different than `widget->class_p` which is the class of the Widget currently being created.

If `LV_OBJ_ID_AUTO_ASSIGN` equates to a non-zero value, `lv_obj_free_id()` is called when a Widget is deleted. It needs to perform the clean-up for any resources allocated by `lv_obj_assign_id()`.

Dumping a Widget Tree

Regardless of the state of any of the above macros, the function `lv_obj_dump_tree(widget)` provides a "dump" of the Widget Tree for the specified Widget (that Widget plus all its children recursively) using the currently-configured method used by the `LV_LOG_USER` macro. If NULL is passed instead of a pointer to a "root" Widget, the dump will include the Widget Tree for all *Screen*, for all *Displays* in the system.

For `LV_LOG_USER` to produce output, the following needs to be true in `lv_conf.h`:

- `LV_USE_LOG` must equate to a non-zero value
- `LV_LOG_LEVEL` <= `LV_LOG_LEVEL_USER`

It will recursively walk through all that Widget's children (starting with the Widget itself) and print the Widget's parent's address, the Widget's address, and if `LV_USE_OBJ_ID` equates to a non-zero value, will also print the output of `lv_obj_stringify_id()` for that Widget.

This can be useful in the event of a UI crash. From that log you can examine the state of the Widget Tree when `lv_obj_dump_tree(widget)` was called.

For example, if a pointer to a deleted Widget is stored in a Timer's `timer->user_data` field when the timer event callback is called, attempted use of that pointer will likely cause a crash because the pointer is not valid any more. However, a timely dump of the Widget Tree right before that point will show that the Widget no longer exists.

Find child by ID

> **Avvertimento**
>
> `lv_obj_find_by_id(widget, id)` is deprecated. To find a widget use `obj_name`.

`lv_obj_find_by_id(widget, id)` will perform a recursive walk through `widget`'s children and return the first child encountered having the given ID.

9.9 Widget Properties

Widget Properties provides a way to greatly reduce the size of the interface between LVGL and whatever logic layer is just above it, to get and set the most important properties of Widgets. It's intended use is to:

- simplify (decreasing development time) writing bindings for LVGL in another language, such as:
  - Micropython,
  - Lua,
  - Python,
  - Perl,
  - .NET
- make it possible to control the UI (or parts of it, e.g. animation) via external input, without modifying firmware, such as:
  - an external text file (YAML, JSON, XML, custom)
  - any external input source (e.g. serial)

While using it consumes more program space and more CPU overhead while setting and getting Widget properties, it is designed so minimize that additional CPU overhead.

9.9.1 What is a Widget Property?

A Widget's properties are the combined set of *Stili* plus additional properties that are unique to each type of Widget, that determine what the Widget looks like and how it behaves. Examples: size, position, color, are properties of all Widgets whereas text, long-mode, selection-start, and selection-end, are properties unique to Label Widgets. A Widget's *local styles* are also valid properties in this context.

The non-style Widget properties available for a given Widget are implemented at the top of that Widget's primary `.c` file as a `const` id-to-function-pointer lookup array, like this example for the Label Widget:

```
#if LV_USE_OBJ_PROPERTY
static const lv_property_ops_t properties[] = {
    {
        .id = LV_PROPERTY_LABEL_TEXT,
        .setter = lv_label_set_text,
        .getter = lv_label_get_text,
    },
    {
        .id = LV_PROPERTY_LABEL_LONG_MODE,
        .setter = lv_label_set_long_mode,
        .getter = lv_label_get_long_mode,
    },
    {
        .id = LV_PROPERTY_LABEL_TEXT_SELECTION_START,
        .setter = lv_label_set_text_selection_start,
        .getter = lv_label_get_text_selection_start,
    },
    {
        .id = LV_PROPERTY_LABEL_TEXT_SELECTION_END,
        .setter = lv_label_set_text_selection_end,
        .getter = lv_label_get_text_selection_end,
    },
};
#endif
```

This array is attached to the `properties` field of the Widget's class, so all Widgets of the same type share the same id-to-function-pointer lookup array.

Some properties are read-only. When this is the case, only the `getter` field in the corresponding array element will be initialized with a function pointer. Example: an object's child-Widget count or scroll position must be controlled via other types of input, but their values are readable through this API.

9.9.2 Utilizzo

By default, this feature of LVGL is turned off. It can be turned on by configuring *`LV_USE_OBJ_PROPERTY`* to `1` in `lv_conf.h`.

The 3 functions that then become available are:

- `lv_result_t lv_obj_set_property(widget, lv_property_t * value)` Sets specified property of Widget.
- `lv_property_t lv_obj_get_property(widget, lv_prop_id_t id)` Reads property value from Widget.
- `lv_result_t lv_obj_set_properties(widget, lv_property_t * values, count)` Sets multiple Widget properties from an array of `lv_property_t`.

An `lv_prop_id_t` is a *Property ID*, whereas an `lv_property_t` is a struct that pairs a *Property ID* with a *Property Value*.

The following is an example of an array that could be used as the `values` argument in `lv_obj_set_properties()`:

```
lv_property_t values[] = {
    { .id = LV_PROPERTY_IMAGE_SRC, .ptr = &img_demo_widgets_avatar, },
    { .id = LV_PROPERTY_IMAGE_PIVOT, .ptr = &pivot_50, },
    { .id = LV_PROPERTY_IMAGE_SCALE, .num = 128, },
    { .id = LV_PROPERTY_OBJ_FLAG_CLICKABLE, .num = 1, },
    { .id = LV_STYLE_IMAGE_OPA, .num = 128, },
    { .id = LV_STYLE_BG_COLOR, .color = (lv_color_t){.red = 0x11, .green =␣
↪0x22, .blue = 0x33}, },
}
```

Alternately, `lv_obj_set_property(widget, value)` could be called using this array's individual `value` elements inside a loop.

Property ID

`lv_prop_id_t` identifies which property to get/set. It is an enum value defined in the primary `.h` file for the Widget in question. Because the actual names are "assembled" by a preprocessor string-concatenation macro and are thus hard to visualize, you can also find the names in the Widget's primary `.c` file in the `properties[]` array initializing the `.id` fields in the array. For example, `LV_PROPERTY_LABEL_TEXT` is one found in `lv_label.c`, and the properties available to all Widgets are found near the top of the `lv_obj.c` file.

That array is attached to the Widget's class, enabling "getter" and "setter" functions to be looked up for each type of Widget where Widget properties has been implemented. (Note: this is done internally so you don't have to.)

If the property you need to set or get using this API is not implemented yet, you can add your own Widget property ID following same rules and using one of two helper macros in the `enum` in the Widget's primary `.h` file. In both cases, the "assembled" value is a 32-bit value:

- `LV_PROPERTY_ID` (for single values -- see *Single Values* below)`; bits `<31:28>` contain the property's value type and bits `<27:0>` contain the property ID.
- `LV_PROPERTY_ID2` (for paired values -- see *Paired Values* below)`; bits `<31:28>` contain the type for the property's 1st value, bits `<27:24>` contain the type for the 2nd value, and bits `<23:0>` contain the property ID.

Just make sure the ID is unique across all Widgets.

Note that `lv_style_prop_t` (enumerator values beginning with `LV_PROPERTY_STYLE_...`) are also valid property IDs, and can be used to set or get a Widget's style values.

Property Value

`lv_property_t` is a struct that begins with an `id` field whose meaning is the same as property ID described above, paired with a value, which is a union of all possible property types including integer, pointer and color. The `value` field is also capable of carrying the different value types for styles. It does this by being a union of all the different types that might be needed. The list of "union-ed" fields at this writing are:

Single Values

```
int32_t             num;      /**< Signed integer number (enums or "normal
↪" numbers) */
uint32_t            num_u;    /**< Unsigned integer number (opacity,␣
↪Booleans) */
bool                enable;   /**< Booleans */
const void        * ptr;      /**< Constant pointers  (font, cone text,␣
↪etc.) */
lv_color_t          color;    /**< Colors */
lv_value_precise_t  precise;  /**< float or int for precise value */
lv_point_t          point;    /**< Point, contains two int32_t */
```

(continues on next page)

(continua dalla pagina precedente)

```
struct {
    /**
     * Note that place struct member `style` at first place is intended.
     * `style` shares same memory with `num`, `ptr`, `color`.
     * So we set the style value directly without using `prop.style.num`.
     *
     * E.g.
     *
     * static const lv_property_t obj_pos_x = {
     *      .id = LV_PROPERTY_STYLE_X,
     *      .num = 123,
     *      .selector = LV_STATE_PRESSED,
     * }
     *
     * instead of:
     * static const lv_property_t obj_pos_x = {
     *      .id = LV_PROPERTY_STYLE_X,
     *      .style.num = 123, // note this line.
     *      .selector = LV_STATE_PRESSED,
     * }
     */
    lv_style_value_t  style;     /**< Make sure it's the first element in␣
↪struct. */
    uint32_t          selector;  /**< Style selector, lv_part_t | lv_state_
↪t */
};
```

Paired Values

```
/**
 * For some properties like slider range, it contains two simple (4-byte)␣
↪values
 * so we can use `arg1.num` and `arg2.num` to set the argument.
 */
struct {
    union {
        int32_t             num;
        uint32_t            num_u;
        bool                enable;
        const void        * ptr;
        lv_color_t          color;
        lv_value_precise_t  precise;
    } arg1, arg2;
};
```

You can find the current `lv_property_t` struct in the lv_obj_property.h file.

Property ID Lookup by Name

Setting configuration macro *LV_USE_OBJ_PROPERTY_NAME* to `1` enables the following functions to look up property IDs by passing property name (a string):

- `lv_prop_id_t lv_obj_property_get_id(widget, name)` Gets property ID by recursively searching for `name` in Widget's class hierarchy, and if still not found, then searches style properties.
- `lv_prop_id_t lv_obj_class_property_get_id(class_p, name)` Gets property ID by doing a non-recursive search for `name` directly in Widget class properties.

- `lv_prop_id_t lv_style_property_get_id(name)` Gets style property ID by name.

The latter two functions are useful when you already know `name` is among the properties of a specific Widget class, or is a style name, since a property name may exist in both lists. Because of the search sequence in `lv_obj_property_get_id(widget, name)`, if a name does exist in both lists, then using this function forces the name in the Widget's class hierarchy properties to have precedence over the style name.

You can tell which names are available by looking in the `.c` files in the `./src/widgets/property/` directory. Note that to support binary name searches, these arrays are generated so that they are guaranteed to be in alphabetical order. If you need to add a property that is not present, it is recommended to add it in the `enum` near the top of the Widget's primary `.h` file, and re-generate these lists using `./scripts/properties.py` to ensure alphabetical ordering is preserved.

9.9.3 Additional Notes

For the `lv_property_t * value` argument of the `lv_obj_set_property()` function, the language used to call that function (e.g. in a static or dynamically-loaded library) may need additional code to convert values from their local data type (e.g. dict, table, etc.) to a C struct before passing it to the `lv_obj_set_property()` function.

9.9.4 API

lv_obj_property_names.h

9.10 Observer

9.10.1 Come si Usa

Panoramica

The `lv_observer` module is an implementation of the Observer Pattern.

This implementation consists of:

Subjects
(in global memory or heap) are "logic packages", each containing the value being "observed" and its type (integer (`int32_t`), a string, a pointer, an *`lv_color_t`*, a `float`, or a group);

Observers
(zero or more per Subject, always dynamically-allocated) are always attached to exactly one Subject, and provide user-defined notifications each the time Subject's value changes.

A Subject and its Observers can be used in various ways:

1. Simply subscribe to a Subject and get notified when the Subject's value changes.
2. Subscribe to a group Subject (connects a group of Subjects) to get notified when any of the Subjects' values change in the group.
3. Bind Widgets to Subjects to automatically match the Widget's value with the Subject (e.g. a Label's text or an Arc's value).

Utilizzo

Using Observer first requires *`LV_USE_OBSERVER`* be configured to `1`. (It is `1` by default, and can be set to `0` to save some program space if you will not be using Observer.)

A typical use case looks like this:

```
// Any typical global variable
lv_subject_t my_subject;

/*-------
```

(continues on next page)

(continua dalla pagina precedente)

```
 * main.c
 *-------*/

extern lv_subject_t my_subject;

void main(void)
{
    // Initialize Subject as integer with the default value of 10.
    lv_subject_init_int(&my_subject, 10);

    some_module_init();
}

/*--------------
 * some_module.c
 *--------------*/

extern lv_subject_t my_subject;

// Will be called when Subject's value changes
static void some_observer_cb(lv_observer_t * observer, lv_subject_t *␣
↪subject)
{
    int32_t v = lv_subject_get_int(subject);
    do_something(v);
}

void some_module_init(void)
{
    // Subscribe to Subject as an Observer.
    lv_subject_add_observer(&my_subject, some_observer_cb, NULL);
}

/*--------------
 * some_system.c
 *--------------*/

extern lv_subject_t my_subject;

void some_event(void)
{
    // The below call sets Subject's value to 30 and notifies current␣
↪Observers.
    lv_subject_set_int(&my_subject, 30);
}
```

Subject

Subject Initialization

Subjects have to be static or global variables, or dynamically-allocated *lv_subject_t* objects. Reason: their content must remain valid through the life of the Subject.

To initialize a Subject use `lv_subject_init_<type>(&subject, params, init_value)`. The following initialization functions exist, one for each of the Subject types:

Integer

void *lv_subject_init_int*(subject, int_value)

Float
void `lv_subject_init_float(subject, float_value)`

String
void `lv_subject_init_string(subject, buf, prev_buf, buf_size, initial_string)`

Pointer
void `lv_subject_init_pointer(subject, ptr)`

Color
void `lv_subject_init_color(subject, color)`

Group
void `lv_subject_init_group(group_subject, subject_list[], count)`

Setting a Subject's Value

The following functions are used to update a Subject's value:

Integer
void `lv_subject_set_int(subject, int_value)`

String
void `lv_subject_copy_string(subject, buf)`

Pointer
void `lv_subject_set_pointer(subject, ptr)`

Color
void `lv_subject_set_color(subject, color)`

At the end of each of these calls, if the new value differs from the previous value, a notification is sent to all current Observers.

Getting a Subject's Value

The following functions are used to get a Subject's current value:

Integer
int32_t `lv_subject_get_int(subject)`

Float
float `lv_subject_get_float(subject)`

String
const char * `lv_subject_get_string(subject)`

Pointer
const void * `lv_subject_get_pointer(subject)`

Color
lv_color_t `lv_subject_get_color(subject)`

Getting a Subject's Previous Value

The following functions are used to get a Subject's previous value:

Integer
int32_t `lv_subject_get_previous_int(subject)`

Float
float `lv_subject_get_previous_float(subject)`

String
const char * `lv_subject_get_previous_string(subject)`

Pointer
const void * `lv_subject_get_previous_pointer(subject)`

Color
lv_color_t `lv_subject_get_previous_color(subject)`

Observer

Subscribing to a Subject

The action of subscribing to a Subject:

- dynamically allocates an Observer object,
- attaches it to the Subject,
- performs an initial notification to the Observer (allowing the Observer to update itself with the Subject's current value), and
- returns a pointer to the newly-created Observer.

Thereafter the Observer will receive a notification each time the Subject's value changes, as long as that Observer remains attached (subscribed) to that Subject.

Notifications are performed by calling the callback function provided when subscribing to the Subject.

Simple Subscription

To subscribe to a Subject one of the `lv_subject_add_observer...()` functions are used. This is covered below.

Alternately, if you want to bind a Subject's value to a Widget's property, one of the `lv_<widget_type>_bind_...()` functions can be used. See *Widget Binding* for more details.

By using `lv_obj_add_subject_...()` it's also possible to change a subject's value on a trigger. It's covered in *Change Subject on Event*

For the most basic use case, subscribe to a Subject by using the following function:

lv_observer_t * observer = `lv_subject_add_observer(&some_subject, some_observer_cb, user_data)`

where the Observer's notification callback should look like this:

```
static void some_observer_cb(lv_observer_t * observer, lv_subject_t *␣
↪subject)
{
    ...
}
```

This function returns a pointer to the newly-created Observer.

When using this method of subscribing, it is the responsibility of the user to call `lv_observer_remove(observer)` when the Observer is no longer needed, which both unsubscribes it from the Subject and deletes it from the LVGL heap.

Subscribing While Associating Observer with a Non-Widget Object

The function subscribes to a Subject additionally associates the Observer with a pointer to any type of object, a copy of which is saved in the Observer's `target` field. This function should be used when the pointer *does not* point to a Widget.

lv_observer_t * observer = `lv_subject_add_observer_with_target(&some_subject, some_observer_cb, some_pointer, user_data)`

A copy of the passed pointer can be retrieved by calling `lv_observer_get_target(observer)`, e.g. inside the callback function.

When using this method of subscribing, it is the responsibility of the user to call `lv_observer_remove(observer)` when the Observer is no longer needed, which both unsubscribes it from the Subject and deletes it from the LVGL heap.

Subscribing While Associating Observer with a Widget

The function below associates a Widget with an Observer while subscribing to a Subject. A copy of the pointer to that Widget is saved in the Observer's `target` field. This works exactly like the above method except that when the Widget is deleted, the Observer thus created will be automatically unsubscribed from the Subject and deleted from the LVGL heap. Note this is different from *Widget Binding*.

```
lv_observer_t * observer = lv_subject_add_observer_obj(&some_subject,
some_observer_cb, widget, user_data)
```

Any number of Observers can be created and be associated with a Widget this way.

A copy of the pointer to the Widget can be retrieved by calling `lv_observer_get_target_obj(observer)`, e.g. inside the callback function. Note that this function returns the stored pointer as a `lv_obj_t *` type, as opposed to the `void *` type returned by `lv_observer_get_target_obj(observer)`. (`lv_observer_get_target(observer)` can still be used if you need that pointer as a `void *` type for any reason, but in practice, this would be rare.)

Unsubscribing from a Subject

When an Observer and its associated events are no longer needed, you can remove it from memory using any of the methods below, depending on the effect you want to create:

- If Widget needs to be deleted, simply delete the Widget, which will automatically gracefully remove the Observer (and its events) from the Subject, while deleting the Widget.
- If Widget does NOT need to be deleted:
  - `lv_obj_remove_from_subject(widget, subject)` deletes all Observers associated with `widget`. `subject` can be NULL to unsubscribe the Widget from all associated Subjects.
  - `lv_subject_deinit(subject)` gracefully disconnects `subject` from all associated Observers and Widget events. This includes subscriptions made using any of the *Widget Binding* functions covered below.
  - `lv_observer_remove(observer)` deletes that specific Observer and gracefully disconnects it from its `subject` and any associated Widgets, where `observer` is the return value any of the above `lv_subject_add_observer_...()` functions.

Subject Groups

When something in your system relies on more than one value (i.e. it needs to be notified when any of a SET of two or more values changes), it can be made an Observer of a Subject Group.

Let us consider an example of an instrument which measures either voltage or current. To display the measured value on a label, 3 things are required:

1. What is being measured (current or voltage)?
2. What is the measured value?
3. What is the range or unit ("mV", "V", "mA", "A")?

When any of these 3 input values change, the label needs to be updated, and it needs to know all 3 values to compose its text.

To handle this you can create an array from the addresses of all the Subjects that are relied upon, and pass that array as a parameter when you initialize a Subject with GROUP type.

```
static lv_subject_t * subject_list[3] = {&subject_1, &subject_2, &subject_
↪3};
lv_subject_init_group(&subject_all, subject_list, 3);  /* Last argument is␣
↪number of elements. */
```

Observers are then added to Subject Groups (e.g. `subject_all`) in the usual way. When this is done, a change to the value of any of the Subjects in the group triggers a notification to all Observers subscribed to the Subject Group (e.g. `subject_all`).

As an example, the above scenario with Voltage/Current measurement might look like this:

```
lv_obj_t * label = lv_label_create(lv_screen_active());

lv_subject_t subject_mode;  // Voltage or Current
lv_subject_t subject_value; // Measured value
lv_subject_t subject_unit;  // The unit
lv_subject_t subject_all;   // Subject group that connects the above 3␣
↪Subjects
lv_subject_t * subject_list[3] = {&subject_mode, &subject_value, &subject_
↪unit};  // The elements of the group

lv_subject_init_int(&subject_mode, 0); // Let's say 0 is Voltage, 1 is␣
↪Current
lv_subject_init_int(&subject_value, 0);
lv_subject_init_pointer(&subject_unit, "V");
lv_subject_init_group(&subject_all, subject_list, 3);

lv_subject_add_observer_obj(&subject_all, all_observer_cb, label, NULL);

...

static void all_observer_cb(lv_observer_t * observer, lv_subject_t *␣
↪subject)
{
    lv_obj_t * label = lv_observer_get_target(observer);
    lv_subject_t * subject_mode = lv_subject_get_group_element(subject, 0);
    lv_subject_t * subject_value = lv_subject_get_group_element(subject,␣
↪1);
    lv_subject_t * subject_unit = lv_subject_get_group_element(subject, 2);

    int32_t mode = lv_subject_get_int(subject_mode);
    int32_t value = lv_subject_get_int(subject_value);
    const char * unit = lv_subject_get_pointer(subject_unit);

    lv_label_set_text_fmt(label, "%s: %d %s", mode ? "Current" : "Voltage",
↪ value, unit);
}
```

Widget Binding

The following methods of subscribing to an integer-type Subject associate the Observer with ONE of a Widget's properties as thought that property itself were the Observer. Any of the following Widget properties can be thus bound to an Subject's integer value:

- flag (or OR-ed combination of flags) from from the `LV_OBJ_FLAG_...` enumeration values;
- state (or OR-ed combination of states) from the `LV_STATE_...` enumeration values;
- text value for Label Widgets;

- integer value for these Widget types:
  - Arco
  - Drop-Down
  - Roller
  - Slider

Any number of Observers can be created for a single Widget, each bound to ONE of the above properties.

For all of the `lv_..._bind_...()` functions covered below, they are similar to `lv_subject_add_observer_obj(&some_subject, some_observer_cb, widget, user_data)` in that they create an Observer and associates the Widget with it. What is different is that updates to the Widget's property thus bound are handled internally -- the user *does not supply callback functions* for any of these subscribing methods -- the callback methods are supplied by the Observer subsystem.

> **Nota**
>
> While the examples below show saving a reference to the created Observer objects for the various `lv_..._bind_...()` functions, it is not necessary to do so unless you need them for some purpose, because the created Observer objects will be automatically deleted when the Widget is deleted.

Any Widget Type

Flags

The following methods of subscribing to an integer Subject affect a Widget's flag (or OR-ed combination of flags). When the subscribing occurs, and each time the Subject's value is changed thereafter, the Subject's value is compared with the specified reference value, and the specified flag(s) is (are):

- SET when the Subject's integer value fulfills the indicated condition, and
- CLEARED otherwise.

Here are the functions that carry out this method of subscribing to a Subject. The `flags` argument can contain a single, or bit-wise OR-ed combination of any of the `LV_OBJ_FLAG_...` enumeration values.

equal
`lv_obj_bind_flag_if_eq(widget, &subject, flags, ref_value)`

not equal
`lv_obj_bind_flag_if_not_eq(widget, &subject, flags, ref_value)`

greater than
`lv_obj_bind_flag_if_gt(widget, &subject, flags, ref_value)`

greater than or equal
`lv_obj_bind_flag_if_ge(widget, &subject, flags, ref_value)`

less than
`lv_obj_bind_flag_if_lt(widget, &subject, flags, ref_value)`

less than or equal
`lv_obj_bind_flag_if_le(widget, &subject, flags, ref_value)`

Stati

The following methods of subscribing to an integer Subject affect a Widget's states (or OR-ed combination of states). When the subscribing occurs, and each time the Subject's value is changed thereafter, the Subject's value is compared with the specified reference value, and the specified state(s) is (are):

- SET when the Subject's integer value fulfills the indicated condition, and

- CLEARED otherwise.

Here are the functions that carry out this method of subscribing to a Subject. The `states` argument can contain a single, or bit-wise OR-ed combination of any of the `LV_STATE_...` enumeration values.

equal
`lv_obj_bind_state_if_eq(widget, &subject, states, ref_value)`

not equal
`lv_obj_bind_state_if_not_eq(widget, &subject, states, ref_value)`

greater than
`lv_obj_bind_state_if_gt(widget, &subject, states, ref_value)`

greater than or equal
`lv_obj_bind_state_if_ge(widget, &subject, states, ref_value)`

less than
`lv_obj_bind_state_if_lt(widget, &subject, states, ref_value)`

less than or equal
`lv_obj_bind_state_if_le(widget, &subject, states, ref_value)`

Checked State

The following method of subscribing to an integer Subject affects a Widget's *LV_STATE_CHECKED* state. When the subscribing occurs, and each time the Subject's value is changed thereafter, the Subject's value is compared to a reference value of `0`, and the *LV_STATE_CHECKED* state is:

- CLEARED when the Subject's value is 0, and
- SET when the Subject's integer value is non-zero.

Note that this is a two-way binding (Subject <===> Widget) so direct (or programmatic) interaction with the Widget that causes its *LV_STATE_CHECKED* state to be SET or CLEARED also causes the Subject's value to be set to `1` or `0` respectively.

- `lv_obj_bind_checked(widget, &subject)`

Label Widgets

This method of subscribing to an integer Subject affects a Label Widget's `text`. The Subject can be an STRING, POINTER or INTEGER type.

When the subscribing occurs, and each time the Subject's value is changed thereafter, the Subject's value is used to update the Label's text as follows:

string Subject
Subject's string is used to directly update the Label's text.

pointer Subject
If NULL is passed as the `format_string` argument when subscribing, the Subject's pointer value is assumed to point to a NUL-terminated string. and is used to directly update the Label's text. See *The format_string Argument* for other options.

integer Subject
Subject's integer value is used with the `format_string` argument. See See *The format_string Argument* for details.

Note that this is a one-way binding (Subject ===> Widget).

- `lv_label_bind_text(label, &subject, format_string)`

The `format_string` Argument

The `format_string` argument is optional and if provided, must contain exactly 1 printf-like format specifier and be one of the following:

string or pointer Subject
"%s" to format the new pointer value as a string or "%p" to format the pointer as a pointer (typically the pointer's address value is spelled out with 4, 8 or 16 hexadecimal characters depending on the platform).

integer Subject
"%d" format specifier (`"%" PRIdxx` --- a cross-platform equivalent where `xx` can be `8`, `16`, `32` or `64`, depending on the platform).

float Subject
"%f" format specifier, or e.g. "%0.2f" to display two digits after the decimal point.

If `NULL` is passed for the `format_string` argument:

string or pointer Subject
Updates expect the pointer to point to a NUL-terminated string.

integer Subject
The Label will simply display the number. Equivalent to "%d".

float Subject
The Label will display the value with "%0.1f" format string.

Example: "%d °C"

Arc Widgets

This method of subscribing to an integer Subject affects an Arc Widget's integer value directly. Note that this is a two-way binding (Subject <===> Widget) so an end user's direct interaction with the Arc Widget updates the Subject's value and vice versa. (Requires *`LV_USE_ARC`* to be configured to `1`.)

It support integer and float subjects.

- *`lv_arc_bind_value`*`(arc, &subject)`

Slider Widgets

This method of subscribing to an integer Subject affects a Slider Widget's integer value directly. Note that this is a two-way binding (Subject <===> Widget) so an end user's direct interaction with the Slider Widget updates the Subject's value and vice versa. (Requires *`LV_USE_SLIDER`* to be configured to `1`.)

It support integer and float subjects.

- *`lv_slider_bind_value`*`(slider, &subject)`

Roller Widgets

This method of subscribing to an integer Subject affects a Roller Widget's integer value directly. Note that this is a two-way binding (Subject <===> Widget) so an end user's direct interaction with the Slider Widget updates the Subject's value and vice versa. (Requires *`LV_USE_ROLLER`* to be configured to `1`.)

It support only integer subjects.

- *`lv_roller_bind_value`*`(roller, &subject)`

Drop-Down Widgets

This method of subscribing to an integer Subject affects a Drop-Down Widget's integer value directly. Note that this is a two-way binding (Subject <===> Widget) so an end user's direct interaction with the Drop-Down Widget updates the Subject's value and vice versa. (Requires `LV_USE_DROPDOWN` to be configured to `1`.)

It support only integer subjects.

- `lv_dropdown_bind_value(dropdown, &subject)`

Change Subject on Event

It's a common requirement to update a subject based on a user action (trigger). To simplify this, *subject set* and *increment* actions can be attached directly to any widget.

Internally, these are implemented as special event callbacks. Note: these callbacks are **not** automatically removed when a subject is deinited.

Increment

`lv_obj_add_subject_increment_event(obj, subject, step, min, max)` Increments the subject's value by `step`, clamped between `min` and `max`.

Ad esempio:

`lv_obj_add_subject_increment_event(button1, subject1, LV_EVENT_CLICKED, 5, -10, 80)`

This will increment `subject1` by 5 when `button1` is clicked. The resulting value will be constrained to the range -10 to 80.

Using a negative `step` will decrement the value instead.

Set to a Value

- `lv_obj_add_subject_set_int_event(obj, subject, trigger, value)`
- `lv_obj_add_subject_set_string_event(obj, subject, trigger, text)`

These functions set the given subject (integer or string) to a fixed value when the specified trigger event occurs.

API

lv_observer.h

lv_observer_private.h

9.10.2 Esempi

Bind a slider's value to a label

```
#include "../../lv_examples.h"
#if LV_USE_OBSERVER && LV_USE_SLIDER && LV_USE_LABEL && LV_BUILD_EXAMPLES

static lv_subject_t temperature_subject;

/**
 * A slider sends a message on value change and a label display's that␣
 ↪value
 */
void lv_example_observer_1(void)
{
    lv_subject_init_int(&temperature_subject, 28);
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Create a slider in the center of the display*/
    lv_obj_t * slider = lv_slider_create(lv_screen_active());
    lv_obj_center(slider);
    lv_slider_bind_value(slider, &temperature_subject);

    /*Create a label below the slider*/
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_align(label, LV_ALIGN_CENTER, 0, 30);
    lv_label_bind_text(label, &temperature_subject, "%d °C");
}

#endif
```

Handling login and its states

```
#include "../../lv_examples.h"
#if LV_USE_OBSERVER && LV_USE_SLIDER && LV_USE_LABEL && LV_BUILD_EXAMPLES

/*This the only interface between the UI and the application*/
static lv_subject_t engine_subject;

static void app_init(void);
static void ui_init(void);

/**
 * Simple PIN login screen to start an engine.
 * The only interface between the UI and the application is a single
 ↪"subject".
 */
void lv_example_observer_2(void)
{
    lv_subject_init_int(&engine_subject, 0);
    app_init();
    ui_init();
}

/*-----------------------------------------------
 * APPLICATION
 *
 * This part contains a demo application logic.
 * It doesn't know anything about the internals of the UI
 * and uses any the `engine_subject` as an interface.
 * -----------------------------------------------*/
static void engine_state_observer_cb(lv_observer_t * observer, lv_subject_
 ↪t * subject)
{
    LV_UNUSED(observer);

    int32_t v = lv_subject_get_int(subject);
    LV_UNUSED(v);
    /*In a real application set/clear a pin here*/
    LV_LOG_USER("Engine state: %" LV_PRId32, v);
}

static void app_init(void)
```

(continues on next page)

(continua dalla pagina precedente)

```
{
    lv_subject_add_observer(&engine_subject, engine_state_observer_cb,␣
↪NULL);
}

/*-----------------------------------------------
 * USER INTERFACE
 *
 * This part contains only UI related code and data.
 * In a project it would a separate file and the
 * application couldn't see its internals
 * -----------------------------------------------*/

typedef enum {
    LOGGED_OUT,
    LOGGED_IN,
    AUTH_FAILED,
} auth_state_t;

static lv_subject_t auth_state_subject;

static void textarea_event_cb(lv_event_t * e)
{
    lv_obj_t * ta = lv_event_get_target_obj(e);
    if(lv_strcmp(lv_textarea_get_text(ta), "hello") == 0) {
        lv_subject_set_int(&auth_state_subject, LOGGED_IN);
    }
    else {
        lv_subject_set_int(&auth_state_subject, AUTH_FAILED);
    }
}

static void info_label_observer_cb(lv_observer_t * observer, lv_subject_t␣
↪* subject)
{
    lv_obj_t * label = (lv_obj_t *) lv_observer_get_target(observer);
    switch(lv_subject_get_int(subject)) {
        case LOGGED_IN:
            lv_label_set_text(label, "Login successful");
            break;
        case LOGGED_OUT:
            lv_label_set_text(label, "Logged out");
            break;
        case AUTH_FAILED:
            lv_label_set_text(label, "Login failed");
            break;
    }
}

static void log_out_click_event_cb(lv_event_t * e)
{
    LV_UNUSED(e);
    lv_subject_set_int(&auth_state_subject, LOGGED_OUT);
}

static void ui_init(void)
```

(continues on next page)

(continua dalla pagina precedente)

```
{
    lv_subject_init_int(&auth_state_subject, LOGGED_OUT);

    /*Create a slider in the center of the display*/
    lv_obj_t * ta = lv_textarea_create(lv_screen_active());
    lv_obj_set_pos(ta, 10, 10);
    lv_obj_set_width(ta, 200);
    lv_textarea_set_one_line(ta, true);
    lv_textarea_set_password_mode(ta, true);
    lv_textarea_set_placeholder_text(ta, "The password is: hello");
    lv_obj_add_event_cb(ta, textarea_event_cb, LV_EVENT_READY, NULL);
    lv_obj_bind_state_if_eq(ta, &auth_state_subject, LV_STATE_DISABLED,␣
↪LOGGED_IN);

    lv_obj_t * kb = lv_keyboard_create(lv_screen_active());
    lv_keyboard_set_textarea(kb, ta);

    lv_obj_t * btn;
    lv_obj_t * label;

    /*Create a log out button which will be active only when logged in*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_set_pos(btn, 220, 10);
    lv_obj_add_event_cb(btn, log_out_click_event_cb, LV_EVENT_CLICKED,␣
↪NULL);
    lv_obj_bind_state_if_not_eq(btn, &auth_state_subject, LV_STATE_
↪DISABLED, LOGGED_IN);

    label = lv_label_create(btn);
    lv_label_set_text(label, "LOG OUT");

    /*Create a label to show info*/
    label = lv_label_create(lv_screen_active());
    lv_obj_set_pos(label, 10, 60);
    lv_subject_add_observer_obj(&auth_state_subject, info_label_observer_
↪cb, label, NULL);

    /*Create button which will be active only when logged in*/
    btn = lv_button_create(lv_screen_active());
    lv_obj_set_pos(btn, 10, 80);
    lv_obj_add_flag(btn, LV_OBJ_FLAG_CHECKABLE);
    lv_obj_bind_state_if_not_eq(btn, &auth_state_subject, LV_STATE_
↪DISABLED, LOGGED_IN);
    lv_button_bind_checked(btn, &engine_subject);
    label = lv_label_create(btn);
    lv_label_set_text(label, "START ENGINE");
}

#endif
```

Set time with 12/24 mode and AM/PM

```
#include "../../lv_examples.h"
#if LV_USE_OBSERVER && LV_USE_SLIDER && LV_USE_LABEL && LV_USE_ROLLER &&␣
↪LV_USE_DROPDOWN && LV_FONT_MONTSERRAT_30 && LV_BUILD_EXAMPLES
```

(continues on next page)

(continua dalla pagina precedente)

```
static lv_subject_t hour_subject;
static lv_subject_t minute_subject;
static lv_subject_t format_subject;
static lv_subject_t am_pm_subject;
static lv_subject_t time_subject;
static lv_subject_t * time_group_array_subject[] = {&hour_subject, &minute_
↪subject, &format_subject, &am_pm_subject};
const char * hour12_options = "01\n02\n03\n04\n05\n06\n07\n08\n09\n10\n11\
↪n12";
const char * hour24_options =
    "00\n01\n02\n03\n04\n05\n06\n07\n08\n09\n10\n11\n12\n13\n14\n15\n16\
↪n17\n18\n19\n20\n21\n22\n23";
const char * minute_options =
    "00\n01\n02\n03\n04\n05\n06\n07\n08\n09\n10\n11\n12\n13\n14\n15\n16\
↪n17\n18\n19\n20\n21\n22\n23\n24\n25\n26\n27\n28\n29\n30\n31\n32\n33\n34\
↪n35\n36\n37\n38\n39\n40\n41\n42\n43\n44\n45\n46\n47\n48\n49\n50\n51\n52\
↪n53\n54\n55\n56\n57\n58\n59";

static void set_btn_clicked_event_cb(lv_event_t * e);
static void close_clicked_event_cb(lv_event_t * e);
static void hour_roller_options_update(lv_observer_t * observer, lv_
↪subject_t * subject);
static void time_observer_cb(lv_observer_t * observer, lv_subject_t *␣
↪subject);

typedef enum {
    TIME_FORMAT_12,
    TIME_FORMAT_24,
} time_format_t;

typedef enum {
    TIME_AM,
    TIME_PM,
} time_am_pm_t;

/**
 * Show how to handle a complex time setting with hour, minute, 12/24 hour␣
↪mode, and AM/PM switch
 * In a real application the time can be displayed on multiple screens and␣
↪it's not trivial
 * how and where to store the current values and how to get them.
 * In this example the widgets to set the time are create/deleted␣
↪dynamically,
 * yet they always know what the current values are by using subjects.
 */
void lv_example_observer_3(void)
{
    /*Initialize the subjects.
     *The UI will update these and read the current values from here,
     *however the application can update these values at any time and
     *the widgets will be updated automatically. */
    lv_subject_init_int(&hour_subject, 7);
    lv_subject_init_int(&minute_subject, 45);
    lv_subject_init_int(&format_subject, TIME_FORMAT_12);
    lv_subject_init_int(&am_pm_subject, TIME_AM);
    lv_subject_init_group(&time_subject, time_group_array_subject, 4);
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*Create the UI*/
    lv_obj_t * time_label = lv_label_create(lv_screen_active());
    lv_obj_set_style_text_font(time_label, &lv_font_montserrat_30, 0);
    lv_subject_add_observer_obj(&time_subject, time_observer_cb, time_
↪label, NULL);
    lv_obj_set_pos(time_label, 24, 24);

    lv_obj_t * set_btn = lv_button_create(lv_screen_active());
    lv_obj_set_pos(set_btn, 180, 24);
    lv_obj_add_event_cb(set_btn, set_btn_clicked_event_cb, LV_EVENT_
↪CLICKED, NULL);

    lv_obj_t * set_label = lv_label_create(set_btn);
    lv_label_set_text(set_label, "Set");

    /*Update some subjects to see if the UI is updated as well*/
    lv_subject_set_int(&hour_subject, 9);
    lv_subject_set_int(&minute_subject, 30);
    lv_subject_set_int(&am_pm_subject, TIME_PM);
}

static void set_btn_clicked_event_cb(lv_event_t * e)
{
    lv_obj_t * set_btn = lv_event_get_target_obj(e);
    lv_obj_add_state(set_btn, LV_STATE_DISABLED);

    lv_obj_t * cont = lv_obj_create(lv_screen_active());
    lv_obj_set_size(cont, lv_pct(100), LV_SIZE_CONTENT);
    lv_obj_align(cont, LV_ALIGN_BOTTOM_MID, 0, 0);

    lv_obj_t * hour_roller = lv_roller_create(cont);
    lv_obj_add_flag(hour_roller, LV_OBJ_FLAG_FLEX_IN_NEW_TRACK);
    lv_subject_add_observer_obj(&format_subject, hour_roller_options_
↪update, hour_roller, NULL);
    lv_roller_bind_value(hour_roller, &hour_subject);
    lv_obj_set_pos(hour_roller, 0, 0);

    lv_obj_t * min_roller = lv_roller_create(cont);
    lv_roller_set_options(min_roller, minute_options, LV_ROLLER_MODE_
↪NORMAL);
    lv_roller_bind_value(min_roller, &minute_subject);
    lv_obj_set_pos(min_roller, 64, 0);

    lv_obj_t * format_dropdown = lv_dropdown_create(cont);
    lv_dropdown_set_options(format_dropdown, "12\n24");
    lv_dropdown_bind_value(format_dropdown, &format_subject);
    lv_obj_set_pos(format_dropdown, 128, 0);
    lv_obj_set_width(format_dropdown, 80);

    lv_obj_t * am_pm_dropdown = lv_dropdown_create(cont);
    lv_dropdown_set_options(am_pm_dropdown, "am\npm");
    lv_dropdown_bind_value(am_pm_dropdown, &am_pm_subject);
    lv_obj_bind_state_if_eq(am_pm_dropdown, &format_subject, LV_STATE_
↪DISABLED, TIME_FORMAT_24);
    lv_obj_set_pos(am_pm_dropdown, 128, 48);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_width(am_pm_dropdown, 80);

    lv_obj_t * close_btn = lv_button_create(cont);
    lv_obj_align(close_btn, LV_ALIGN_TOP_RIGHT, 0, 0);
    /*Pass the set_btn as user_data to make it non-disabled on close*/
    lv_obj_add_event_cb(close_btn, close_clicked_event_cb, LV_EVENT_
↪CLICKED, set_btn);

    lv_obj_t * close_label = lv_label_create(close_btn);
    lv_label_set_text(close_label, LV_SYMBOL_CLOSE);
}

static void close_clicked_event_cb(lv_event_t * e)
{
    lv_obj_t * set_btn = (lv_obj_t *) lv_event_get_user_data(e);
    lv_obj_t * close_btn = lv_event_get_target_obj(e);
    lv_obj_t * cont = lv_obj_get_parent(close_btn);
    lv_obj_remove_state(set_btn, LV_STATE_DISABLED);
    lv_obj_delete(cont);
}

/*Watch all related subject to display the current time correctly*/
static void time_observer_cb(lv_observer_t * observer, lv_subject_t *␣
↪subject)
{
    int32_t hour = lv_subject_get_int(lv_subject_get_group_element(subject,
↪ 0));
    int32_t minute = lv_subject_get_int(lv_subject_get_group_
↪element(subject, 1));
    int32_t format = lv_subject_get_int(lv_subject_get_group_
↪element(subject, 2));
    int32_t am_pm = lv_subject_get_int(lv_subject_get_group_
↪element(subject, 3));

    lv_obj_t * label = (lv_obj_t *) lv_observer_get_target(observer);

    if(format == TIME_FORMAT_24) {
        lv_label_set_text_fmt(label, "%" LV_PRId32 ":%02" LV_PRId32, hour,␣
↪minute);
    }
    else {
        lv_label_set_text_fmt(label, "%" LV_PRId32":%02" LV_PRId32" %s",␣
↪hour + 1, minute, am_pm == TIME_AM ? "am" : "pm");
    }
}

/*Change the hour options on format change*/
static void hour_roller_options_update(lv_observer_t * observer, lv_
↪subject_t * subject)
{
    lv_obj_t * roller = (lv_obj_t *) lv_observer_get_target(observer);
    int32_t prev_selected = lv_roller_get_selected(roller);
    int32_t v = lv_subject_get_int(subject);
    if(v == TIME_FORMAT_12) {
        prev_selected--;
        if(prev_selected > 12) prev_selected -= 12;
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_roller_set_options(roller, hour12_options, LV_ROLLER_MODE_
↪NORMAL);
    }
    else {
        prev_selected++;
        lv_roller_set_options(roller, hour24_options, LV_ROLLER_MODE_
↪NORMAL);
    }

    lv_roller_set_selected(roller, prev_selected, LV_ANIM_OFF);
    lv_obj_send_event(roller, LV_EVENT_VALUE_CHANGED, NULL);
}

#endif
```

Custom tab view with state management

```
#include "../../lv_examples.h"
#if LV_USE_OBSERVER && LV_USE_SLIDER && LV_USE_LABEL && LV_USE_ROLLER &&␣
↪LV_USE_DROPDOWN && LV_FONT_MONTSERRAT_30 && LV_BUILD_EXAMPLES

static void cont_observer_cb(lv_observer_t * observer, lv_subject_t *␣
↪subject);
static void btn_create(lv_obj_t * parent, const char * text);
static void btn_click_event_cb(lv_event_t * e);
static void btn_observer_cb(lv_observer_t * observer, lv_subject_t *␣
↪subject);
static void indicator_observer_cb(lv_observer_t * observer, lv_subject_t *␣
↪subject);

static lv_subject_t current_tab_subject;
static lv_subject_t slider_subject[4];
static lv_subject_t dropdown_subject[3];
static lv_subject_t roller_subject[2];

void lv_example_observer_4(void)
{
    lv_subject_init_int(&current_tab_subject, 0);
    lv_subject_init_int(&slider_subject[0], 0);
    lv_subject_init_int(&slider_subject[1], 0);
    lv_subject_init_int(&slider_subject[2], 0);
    lv_subject_init_int(&slider_subject[3], 0);
    lv_subject_init_int(&dropdown_subject[0], 0);
    lv_subject_init_int(&dropdown_subject[1], 0);
    lv_subject_init_int(&dropdown_subject[2], 0);
    lv_subject_init_int(&roller_subject[0], 0);
    lv_subject_init_int(&roller_subject[1], 0);

    lv_obj_t * main_cont = lv_obj_create(lv_screen_active());
    lv_obj_remove_style_all(main_cont);
    lv_obj_set_size(main_cont, lv_pct(100), lv_pct(100));
    lv_obj_set_style_pad_all(main_cont, 0, 0);
    lv_obj_set_flex_flow(main_cont, LV_FLEX_FLOW_COLUMN);

    lv_obj_t * cont = lv_obj_create(main_cont);
    lv_obj_remove_style_all(cont);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_flex_grow(cont, 1);
    lv_obj_set_style_pad_all(cont, 8, 0);
    lv_obj_set_width(cont, lv_pct(100));
    lv_subject_add_observer_obj(&current_tab_subject, cont_observer_cb,␣
↪cont, NULL);
    lv_obj_set_scroll_dir(cont, LV_DIR_VER);

    lv_obj_t * footer = lv_obj_create(main_cont);
    lv_obj_remove_style_all(footer);
    lv_obj_set_style_pad_column(footer, 8, 0);
    lv_obj_set_style_pad_all(footer, 8, 0);
    lv_obj_set_flex_flow(footer, LV_FLEX_FLOW_ROW);
    lv_obj_set_size(footer, lv_pct(100), 60);
    lv_obj_align(footer, LV_ALIGN_BOTTOM_MID, 0, 0);

    btn_create(footer, "First");
    btn_create(footer, "Second");
    btn_create(footer, "Third");

    lv_obj_t * indicator = lv_obj_create(footer);
    lv_obj_remove_style(indicator, NULL, 0);
    lv_obj_set_style_bg_opa(indicator, LV_OPA_40, 0);
    lv_subject_add_observer_obj(&current_tab_subject, indicator_observer_
↪cb, indicator, NULL);
    lv_obj_set_height(indicator, 10);
    lv_obj_align(indicator, LV_ALIGN_BOTTOM_LEFT, 0, 0);
    lv_obj_add_flag(indicator, LV_OBJ_FLAG_IGNORE_LAYOUT);

    /*Be sure the indicator has the correct size*/
    lv_obj_update_layout(indicator);
    lv_subject_notify(&current_tab_subject);
}

static int32_t anim_get_x_cb(lv_anim_t * a)
{
    return lv_obj_get_x_aligned((lv_obj_t *) a->var);
}

static void anim_set_x_cb(void * obj, int32_t v)
{
    lv_obj_set_x((lv_obj_t *) obj, v);
}

static void cont_observer_cb(lv_observer_t * observer, lv_subject_t *␣
↪subject)
{
    int32_t prev_v = lv_subject_get_previous_int(subject);
    int32_t cur_v = lv_subject_get_int(subject);
    lv_obj_t * cont = (lv_obj_t *) lv_observer_get_target(observer);

    /*Animate out the previous content*/
    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_duration(&a, 300);
    lv_anim_set_path_cb(&a, lv_anim_path_ease_in_out);
    lv_anim_set_exec_cb(&a, anim_set_x_cb);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_anim_set_get_value_cb(&a, anim_get_x_cb);
    lv_anim_set_completed_cb(&a, lv_obj_delete_anim_completed_cb);

    uint32_t i;
    uint32_t delay = 0;
    uint32_t child_cnt_prev = lv_obj_get_child_count(cont);
    for(i = 0; i < child_cnt_prev; i++) {
        lv_obj_t * child = lv_obj_get_child(cont, i);
        lv_anim_set_var(&a, child);
        if(prev_v < cur_v) {
            lv_anim_set_values(&a, 0, -20);
        }
        else {
            lv_anim_set_values(&a, 0, 20);
        }
        lv_anim_set_delay(&a, delay);
        lv_anim_start(&a);
        lv_obj_fade_out(child, 200, delay);
        delay += 50;
    }

    /*Create the widgets according to the current value*/
    if(cur_v == 0) {
        for(i = 0; i < 4; i++) {
            lv_obj_t * slider = lv_slider_create(cont);
            lv_slider_bind_value(slider, &slider_subject[i]);
            lv_obj_align(slider, LV_ALIGN_TOP_MID, 0, 10 + i * 30);
        }
    }
    if(cur_v == 1) {
        for(i = 0; i < 3; i++) {
            lv_obj_t * dropdown = lv_dropdown_create(cont);
            lv_dropdown_bind_value(dropdown, &dropdown_subject[i]);
            lv_obj_align(dropdown, LV_ALIGN_TOP_MID, 0, i * 50);
        }
    }
    if(cur_v == 2) {
        for(i = 0; i < 2; i++) {
            lv_obj_t * roller = lv_roller_create(cont);
            lv_roller_bind_value(roller, &roller_subject[i]);
            lv_obj_align(roller, LV_ALIGN_CENTER, - 80 + i * 160, 0);
        }
    }

    /*Animate in the new widgets*/
    lv_anim_set_completed_cb(&a, NULL);
    for(i = child_cnt_prev; i < lv_obj_get_child_count(cont); i++) {
        lv_obj_t * child = lv_obj_get_child(cont, i);
        lv_anim_set_var(&a, child);
        if(prev_v < cur_v) {
            lv_anim_set_values(&a, 20, 0);
        }
        else {
            lv_anim_set_values(&a, -20, -0);
        }
        lv_anim_set_delay(&a, delay);
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_anim_start(&a);
        lv_obj_fade_in(child, 200, delay);
        delay += 50;
    }

}

static void btn_create(lv_obj_t * parent, const char * text)
{
    lv_obj_t * btn = lv_button_create(parent);
    lv_obj_set_flex_grow(btn, 1);
    lv_obj_set_height(btn, lv_pct(100));
    lv_obj_set_style_radius(btn, 0, 0);
    lv_subject_add_observer_obj(&current_tab_subject, btn_observer_cb, btn,
 ↪ NULL);
    lv_obj_add_event_cb(btn, btn_click_event_cb, LV_EVENT_CLICKED, NULL);

    lv_obj_t * label = lv_label_create(btn);
    lv_label_set_text(label, text);
    lv_obj_center(label);
}

static void btn_click_event_cb(lv_event_t * e)
{
    lv_obj_t * btn = lv_event_get_target_obj(e);
    uint32_t idx = lv_obj_get_index(btn);
    lv_subject_set_int(&current_tab_subject, idx);
}

static void btn_observer_cb(lv_observer_t * observer, lv_subject_t *␣
 ↪subject)
{
    int32_t prev_v = lv_subject_get_previous_int(subject);
    int32_t cur_v = lv_subject_get_int(subject);

    lv_obj_t * btn = (lv_obj_t *) lv_observer_get_target(observer);
    int32_t idx = (int32_t)lv_obj_get_index(btn);

    if(idx == prev_v) lv_obj_remove_state(btn, LV_STATE_CHECKED);
    if(idx == cur_v) lv_obj_add_state(btn, LV_STATE_CHECKED);
}

static void indicator_observer_cb(lv_observer_t * observer, lv_subject_t *␣
 ↪subject)
{
    int32_t cur_v = lv_subject_get_int(subject);
    lv_obj_t * indicator = (lv_obj_t *) lv_observer_get_target(observer);

    lv_obj_t * footer = lv_obj_get_parent(indicator);
    lv_obj_t * btn_act = lv_obj_get_child(footer, cur_v);
    lv_obj_set_width(indicator, lv_obj_get_width(btn_act));

    lv_anim_t a;
    lv_anim_init(&a);
    lv_anim_set_exec_cb(&a, anim_set_x_cb);
    lv_anim_set_duration(&a, 300);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_anim_set_path_cb(&a, lv_anim_path_ease_in_out);
    lv_anim_set_var(&a, indicator);
    lv_anim_set_values(&a, lv_obj_get_x(indicator), lv_obj_get_x(btn_act));
    lv_anim_start(&a);

}

#endif
```

Firmware update process

```
#include "../../lv_examples.h"
#if LV_USE_OBSERVER && LV_USE_ARC && LV_USE_LABEL && LV_USE_BUTTON && LV_
↪USE_SPINNER && LV_BUILD_EXAMPLES

typedef enum {
    FW_UPDATE_STATE_IDLE,
    FW_UPDATE_STATE_CONNECTING,
    FW_UPDATE_STATE_CONNECTED,
    FW_UPDATE_STATE_DOWNLOADING,
    FW_UPDATE_STATE_CANCEL,
    FW_UPDATE_STATE_READY,
} lv_fw_update_state_t;

static void fw_upload_manager_observer_cb(lv_observer_t * observer, lv_
↪subject_t * subject);
static void fw_update_btn_clicked_event_cb(lv_event_t * e);
static void fw_update_close_event_cb(lv_event_t * e);
static void fw_update_win_observer_cb(lv_observer_t * observer, lv_subject_
↪t * subject);

static lv_subject_t fw_download_percent_subject;
static lv_subject_t fw_update_status_subject;

/**
 * Show how to handle a complete firmware update process with observers.
 * Normally it's hard to implement a firmware update process because in␣
↪some cases
 *   - the App needs to was for the UI (wait for a button press)
 *   - the UI needs to wait for the App (connecting or downloading)
 * With observers these complex mechanisms can be implemented a simple and␣
↪clean way.
 */
void lv_example_observer_5(void)
{
    lv_subject_init_int(&fw_download_percent_subject, 0);
    lv_subject_init_int(&fw_update_status_subject, FW_UPDATE_STATE_IDLE);

    lv_subject_add_observer(&fw_update_status_subject, fw_upload_manager_
↪observer_cb, NULL);

    /*Create start FW update button*/
    lv_obj_t * btn = lv_button_create(lv_screen_active());
    lv_obj_add_event_cb(btn, fw_update_btn_clicked_event_cb, LV_EVENT_
↪CLICKED, NULL);
    lv_obj_center(btn);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_t * label = lv_label_create(btn);
    lv_label_set_text(label, "Firmware update");
}

static void fw_update_btn_clicked_event_cb(lv_event_t * e)
{
    LV_UNUSED(e);
    lv_obj_t * win = lv_win_create(lv_screen_active());
    lv_obj_set_size(win, lv_pct(90), lv_pct(90));
    lv_obj_set_height(lv_win_get_header(win), 40);
    lv_obj_set_style_radius(win, 8, 0);
    lv_obj_set_style_shadow_width(win, 24, 0);
    lv_obj_set_style_shadow_offset_x(win, 2, 0);
    lv_obj_set_style_shadow_offset_y(win, 3, 0);
    lv_obj_set_style_shadow_color(win, lv_color_hex3(0x888), 0);
    lv_win_add_title(win, "Firmware update");
    lv_obj_t * btn = lv_win_add_button(win, LV_SYMBOL_CLOSE, 40);
    lv_obj_add_event_cb(btn, fw_update_close_event_cb, LV_EVENT_CLICKED,␣
↪NULL);
    lv_obj_center(win);

    lv_subject_set_int(&fw_update_status_subject, FW_UPDATE_STATE_IDLE);
    lv_subject_add_observer_obj(&fw_update_status_subject, fw_update_win_
↪observer_cb, win, NULL);
}

static void fw_update_close_event_cb(lv_event_t * e)
{
    LV_UNUSED(e);
    lv_subject_set_int(&fw_update_status_subject, FW_UPDATE_STATE_CANCEL);
}

static void restart_btn_click_event_cb(lv_event_t * e)
{
    lv_obj_t * win = (lv_obj_t *) lv_event_get_user_data(e);
    lv_obj_delete(win);
    lv_subject_set_int(&fw_update_status_subject, FW_UPDATE_STATE_IDLE);
}

static void fw_update_win_observer_cb(lv_observer_t * observer, lv_subject_
↪t * subject)
{
    lv_obj_t * win = (lv_obj_t *) lv_observer_get_target(observer);
    lv_obj_t * cont = lv_win_get_content(win);
    lv_fw_update_state_t status = (lv_fw_update_state_t) lv_subject_get_
↪int(&fw_update_status_subject);
    if(status == FW_UPDATE_STATE_IDLE) {
        lv_obj_clean(cont);
        lv_obj_t * spinner = lv_spinner_create(cont);
        lv_obj_center(spinner);
        lv_obj_set_size(spinner, 130, 130);

        lv_obj_t * label = lv_label_create(cont);
        lv_label_set_text(label, "Connecting");
        lv_obj_center(label);
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_subject_set_int(subject, FW_UPDATE_STATE_CONNECTING);
    }
    else if(status == FW_UPDATE_STATE_DOWNLOADING) {
        lv_obj_clean(cont);
        lv_obj_t * arc = lv_arc_create(cont);
        lv_arc_bind_value(arc, &fw_download_percent_subject);
        lv_obj_center(arc);
        lv_obj_set_size(arc, 130, 130);
        lv_obj_remove_flag(arc, LV_OBJ_FLAG_CLICKABLE);

        lv_obj_t * label = lv_label_create(cont);
        lv_label_bind_text(label, &fw_download_percent_subject, "%d %%");
        lv_obj_center(label);
    }
    else if(status == FW_UPDATE_STATE_READY) {
        lv_obj_clean(cont);
        lv_obj_t * label = lv_label_create(cont);
        lv_label_set_text(label, "Firmware update is ready");
        lv_obj_align(label, LV_ALIGN_CENTER, 0, -20);

        lv_obj_t * btn = lv_button_create(cont);
        lv_obj_align(btn, LV_ALIGN_CENTER, 0, 20);
        lv_obj_add_event_cb(btn, restart_btn_click_event_cb, LV_EVENT_
↪CLICKED, win);

        label = lv_label_create(btn);
        lv_label_set_text(label, "Restart");
    }
    else if(status == FW_UPDATE_STATE_CANCEL) {
        lv_obj_delete(win);
    }
}

static void connect_timer_cb(lv_timer_t * t)
{
    if(lv_subject_get_int(&fw_update_status_subject) != FW_UPDATE_STATE_
↪CANCEL) {
        lv_subject_set_int(&fw_update_status_subject, FW_UPDATE_STATE_
↪CONNECTED);
    }
    lv_timer_delete(t);
}

static void download_timer_cb(lv_timer_t * t)
{
    if(lv_subject_get_int(&fw_update_status_subject) == FW_UPDATE_STATE_
↪CANCEL) {
        lv_timer_delete(t);
        return;
    }

    int32_t v = lv_subject_get_int(&fw_download_percent_subject);
    if(v < 100) {
        lv_subject_set_int(&fw_download_percent_subject, v + 1);
    }
    else {
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_subject_set_int(&fw_update_status_subject, FW_UPDATE_STATE_
↪READY);
        lv_timer_delete(t);
    }
}

/**
 * Emulate connection and FW downloading by timers
 */
static void fw_upload_manager_observer_cb(lv_observer_t * observer, lv_
↪subject_t * subject)
{
    LV_UNUSED(subject);
    LV_UNUSED(observer);

    lv_fw_update_state_t state = (lv_fw_update_state_t) lv_subject_get_
↪int(&fw_update_status_subject);
    if(state == FW_UPDATE_STATE_CONNECTING) {
        lv_timer_create(connect_timer_cb, 2000, NULL);
    }
    else if(state == FW_UPDATE_STATE_CONNECTED) {
        lv_subject_set_int(&fw_download_percent_subject, 0);
        lv_subject_set_int(&fw_update_status_subject, FW_UPDATE_STATE_
↪DOWNLOADING);
        lv_timer_create(download_timer_cb, 50, NULL);
    }
}

#endif
```

Modular style update on theme change

```
#include "../../lv_examples.h"
#if LV_USE_OBSERVER && LV_USE_SLIDER && LV_USE_LABEL && LV_BUILD_EXAMPLES

typedef enum {
    THEME_MODE_LIGHT,
    THEME_MODE_DARK,
} lv_theme_mode_t;

static lv_obj_t * my_panel_create(lv_obj_t * parent);
static lv_obj_t * my_button_create(lv_obj_t * parent, const char * text,␣
↪lv_event_cb_t event_cb);
static void switch_theme_event_cb(lv_event_t * e);

static lv_subject_t theme_subject;

/**
 * Change between light and dark mode
 */
void lv_example_observer_6(void)
{
    lv_subject_init_int(&theme_subject, THEME_MODE_DARK);

    lv_obj_t * panel1 = my_panel_create(lv_screen_active());
    lv_obj_set_flex_flow(panel1, LV_FLEX_FLOW_COLUMN);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_set_flex_align(panel1, LV_FLEX_ALIGN_START, LV_FLEX_ALIGN_
↪CENTER, LV_FLEX_ALIGN_CENTER);
    lv_obj_set_size(panel1, lv_pct(90), lv_pct(90));
    lv_obj_center(panel1);

    my_button_create(panel1, "Button 1", switch_theme_event_cb);
    my_button_create(panel1, "Button 2", switch_theme_event_cb);
    my_button_create(panel1, "Button 3", switch_theme_event_cb);
    my_button_create(panel1, "Button 4", switch_theme_event_cb);
    my_button_create(panel1, "Button 5", switch_theme_event_cb);
    my_button_create(panel1, "Button 6", switch_theme_event_cb);
    my_button_create(panel1, "Button 7", switch_theme_event_cb);
    my_button_create(panel1, "Button 8", switch_theme_event_cb);
    my_button_create(panel1, "Button 9", switch_theme_event_cb);
    my_button_create(panel1, "Button 10", switch_theme_event_cb);
}

static void switch_theme_event_cb(lv_event_t * e)
{
    LV_UNUSED(e);
    if(lv_subject_get_int(&theme_subject) == THEME_MODE_LIGHT) lv_subject_
↪set_int(&theme_subject, THEME_MODE_DARK);
    else lv_subject_set_int(&theme_subject, THEME_MODE_LIGHT);
}

/*-----------------------------------------
 * my_panel.c
 *
 * It would be a separate file with its own
 * local types, data, and functions
 *-----------------------------------------*/

typedef struct {
    lv_style_t style_main;
    lv_style_t style_scrollbar;
} lv_panel_styles_t;

static void my_panel_style_observer_cb(lv_observer_t * observer, lv_
↪subject_t * subject)
{
    LV_UNUSED(subject);
    LV_UNUSED(observer);

    lv_theme_mode_t m = (lv_theme_mode_t) lv_subject_get_int(&theme_
↪subject);
    lv_panel_styles_t * styles = (lv_panel_styles_t *) lv_observer_get_
↪target(observer);
    if(m == THEME_MODE_LIGHT) {
        lv_style_set_bg_color(&styles->style_main, lv_color_hex3(0xfff));
        lv_style_set_shadow_color(&styles->style_main, lv_color_
↪hex3(0x888));
        lv_style_set_text_color(&styles->style_main, lv_color_hex3(0x222));
        lv_style_set_bg_color(&styles->style_scrollbar, lv_color_
↪hex3(0x888));
    }
    if(m == THEME_MODE_DARK) {
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_style_set_bg_color(&styles->style_main, lv_color_hex(0x040038));
        lv_style_set_shadow_color(&styles->style_main, lv_color_
↪hex3(0xaaa));
        lv_style_set_text_color(&styles->style_main, lv_color_hex3(0xeee));
        lv_style_set_bg_color(&styles->style_scrollbar, lv_color_
↪hex3(0xaaa));
    }

    lv_obj_report_style_change(&styles->style_main);
    lv_obj_report_style_change(&styles->style_scrollbar);
}

static lv_obj_t * my_panel_create(lv_obj_t * parent)
{
    static bool inited = false;
    static lv_panel_styles_t styles;
    if(!inited) {
        inited = true;

        lv_style_init(&styles.style_main);
        lv_style_set_radius(&styles.style_main, 12);
        lv_style_set_bg_opa(&styles.style_main, LV_OPA_COVER);
        lv_style_set_shadow_width(&styles.style_main, 24);
        lv_style_set_shadow_offset_x(&styles.style_main, 4);
        lv_style_set_shadow_offset_y(&styles.style_main, 6);
        lv_style_set_pad_all(&styles.style_main, 12);
        lv_style_set_pad_gap(&styles.style_main, 16);

        lv_style_init(&styles.style_scrollbar);
        lv_style_set_width(&styles.style_scrollbar, 4);
        lv_style_set_radius(&styles.style_scrollbar, 2);
        lv_style_set_pad_right(&styles.style_scrollbar, 8);
        lv_style_set_pad_ver(&styles.style_scrollbar, 8);
        lv_style_set_bg_opa(&styles.style_scrollbar, LV_OPA_50);

        lv_subject_add_observer_with_target(&theme_subject, my_panel_style_
↪observer_cb, &styles, NULL);
    }

    lv_obj_t * panel = lv_obj_create(parent);
    lv_obj_remove_style_all(panel);
    lv_obj_add_style(panel, &styles.style_main, 0);
    lv_obj_add_style(panel, &styles.style_scrollbar, LV_PART_SCROLLBAR);

    return panel;
}

/*-----------------------------------------
 * my_button.c
 *
 * It would be a separate file with its own
 * local types, data, and functions
 *-----------------------------------------*/

typedef struct {
    lv_style_t style_main;
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_t style_pressed;
} lv_button_styles_t;

static void my_button_style_observer_cb(lv_observer_t * observer, lv_
↪subject_t * subject)
{
    LV_UNUSED(subject);
    LV_UNUSED(observer);

    lv_theme_mode_t m = (lv_theme_mode_t) lv_subject_get_int(&theme_
↪subject);
    lv_button_styles_t * styles = (lv_button_styles_t *) lv_observer_get_
↪target(observer);
    if(m == THEME_MODE_LIGHT) {
        lv_style_set_bg_color(&styles->style_main, lv_color_hex(0x3379de));
        lv_style_set_bg_grad_color(&styles->style_main, lv_color_
↪hex(0xd249a5));
        lv_style_set_shadow_color(&styles->style_main, lv_color_
↪hex(0x3379de));
        lv_style_set_text_color(&styles->style_main, lv_color_hex3(0xfff));

        lv_style_set_color_filter_opa(&styles->style_pressed, LV_OPA_70);
    }
    if(m == THEME_MODE_DARK) {
        lv_style_set_bg_color(&styles->style_main, lv_color_hex(0xde1382));
        lv_style_set_bg_grad_color(&styles->style_main, lv_color_
↪hex(0x4b0c72));
        lv_style_set_shadow_color(&styles->style_main, lv_color_
↪hex(0x4b0c72));
        lv_style_set_text_color(&styles->style_main, lv_color_hex3(0xfff));

        lv_style_set_color_filter_opa(&styles->style_pressed, LV_OPA_30);
    }

    lv_obj_report_style_change(&styles->style_main);
    lv_obj_report_style_change(&styles->style_pressed);
}

static lv_obj_t * my_button_create(lv_obj_t * parent, const char * text,␣
↪lv_event_cb_t event_cb)
{
    static bool inited = false;
    static lv_button_styles_t styles;
    if(!inited) {
        inited = true;

        lv_style_init(&styles.style_main);
        lv_style_set_radius(&styles.style_main, LV_RADIUS_CIRCLE);
        lv_style_set_bg_opa(&styles.style_main, LV_OPA_COVER);
        lv_style_set_bg_grad_dir(&styles.style_main, LV_GRAD_DIR_HOR);
        lv_style_set_shadow_width(&styles.style_main, 24);
        lv_style_set_shadow_offset_y(&styles.style_main, 6);
        lv_style_set_pad_hor(&styles.style_main, 32);
        lv_style_set_pad_ver(&styles.style_main, 12);

        lv_style_init(&styles.style_pressed);
```

(continues on next page)

(continua dalla pagina precedente)

```
        lv_style_set_color_filter_dsc(&styles.style_pressed, &lv_color_
↪filter_shade);
        lv_subject_add_observer_with_target(&theme_subject, my_button_
↪style_observer_cb, &styles, NULL);
    }

    lv_obj_t * btn = lv_button_create(parent);
    lv_obj_remove_style_all(btn);
    lv_obj_add_style(btn, &styles.style_main, 0);
    lv_obj_add_style(btn, &styles.style_pressed, LV_STATE_PRESSED);
    lv_obj_add_event_cb(btn, event_cb, LV_EVENT_CLICKED, NULL);

    lv_obj_t * label = lv_label_create(btn);
    lv_label_set_text(label, text);

    return btn;
}

#endif
```

9.11 Snapshot

Snapshot provides an API to take a snapshot image for an LVGL Widget together with its children. The image will look exactly like the Widget on the display.

9.11.1 Utilizzo

Simply call function `lv_snapshot_take(widget, color_format)` to generate the image descriptor which can be used as an Image Widget's image source using `lv_image_set_src()`.

Note, only following color formats are supported at this time:

- *LV_COLOR_FORMAT_RGB565*
- *LV_COLOR_FORMAT_RGB888*
- *LV_COLOR_FORMAT_XRGB8888*
- *LV_COLOR_FORMAT_ARGB8888*

Freeing the Image

The memory `lv_snapshot_take()` uses is dynamically allocated using `lv_draw_buf_create()`. Use `lv_draw_buf_destroy()` to free the memory it allocated.

The snapshot image which is the draw buffer returned by `lv_snapshot_take()` normally won't be added to the cache because it can be drawn directly. So you don't need to invalidate the cache by calling `lv_image_cache_drop()` before destroying the draw buffer.

The below code snippet demonstrates correct use of `lv_snapshot_take()`:

```
void update_snapshot(lv_obj_t * widget, lv_obj_t * img_snapshot)
{
    lv_draw_buf_t* snapshot = (void*)lv_image_get_src(img_snapshot);
    if(snapshot) {
        lv_draw_buf_destroy(snapshot);
    }
    snapshot = lv_snapshot_take(widget, LV_COLOR_FORMAT_ARGB8888);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_image_set_src(img_snapshot, snapshot);
}
```

Using an Existing Buffer

If the snapshot needs to be updated repeatedly, or if the caller provides the draw buffer, use `lv_snapshot_take_to_draw_buf(widget, color_format, draw_buf)`. In this case, the caller is responsible for creating and destroying the draw buffer.

If snapshot is generated successfully, the image descriptor is updated, the image data will be stored to the provided `draw_buf`, and the function will return `LV_RESULT_OK`.

Note that snapshot may fail if the provided buffer is not large enough, which can happen if the Widget's size changes. It's recommended to use `lv_snapshot_reshape_draw_buf(widget, draw_buf)` to first ensure the buffer is large enough, and if it fails, destroy the existing draw buffer and call `lv_snapshot_take` directly.

9.11.2 Esempio

Simple snapshot example

```
#include "../../lv_examples.h"
#if LV_USE_SNAPSHOT && LV_BUILD_EXAMPLES

static void event_cb(lv_event_t * e)
{
    lv_obj_t * snapshot_obj = (lv_obj_t *) lv_event_get_user_data(e);
    lv_obj_t * img = lv_event_get_target_obj(e);

    if(snapshot_obj) {
        lv_draw_buf_t * snapshot = (lv_draw_buf_t *)lv_image_get_
↪src(snapshot_obj);
        if(snapshot) {
            lv_draw_buf_destroy(snapshot);
        }

        /*Update the snapshot, we know parent of object is the container.*/
        snapshot = lv_snapshot_take(lv_obj_get_parent(img), LV_COLOR_
↪FORMAT_ARGB8888);
        if(snapshot == NULL)
            return;
        lv_image_set_src(snapshot_obj, snapshot);
    }
}

void lv_example_snapshot_1(void)
{
    LV_IMAGE_DECLARE(img_star);
    lv_obj_t * root = lv_screen_active();
    lv_obj_set_style_bg_color(root, lv_palette_main(LV_PALETTE_LIGHT_BLUE),
↪ 0);

    /*Create an image object to show snapshot*/
    lv_obj_t * snapshot_obj = lv_image_create(root);
    lv_obj_set_style_bg_color(snapshot_obj, lv_palette_main(LV_PALETTE_
↪PURPLE), 0);
    lv_obj_set_style_bg_opa(snapshot_obj, LV_OPA_100, 0);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_image_set_scale(snapshot_obj, 128);
    lv_image_set_rotation(snapshot_obj, 300);

    /*Create the container and its children*/
    lv_obj_t * container = lv_obj_create(root);

    lv_obj_center(container);
    lv_obj_set_size(container, 180, 180);
    lv_obj_set_flex_flow(container, LV_FLEX_FLOW_ROW_WRAP);
    lv_obj_set_flex_align(container, LV_FLEX_ALIGN_SPACE_EVENLY, LV_FLEX_
↪ALIGN_CENTER, LV_FLEX_ALIGN_CENTER);
    lv_obj_set_style_radius(container, 50, 0);
    lv_obj_t * img;
    int i;
    for(i = 0; i < 4; i++) {
        img = lv_image_create(container);
        lv_image_set_src(img, &img_star);
        lv_obj_set_style_bg_color(img, lv_color_black(), 0);
        lv_obj_set_style_bg_opa(img, LV_OPA_COVER, 0);
        //        lv_obj_set_style_transform_scale(img, 400, LV_STATE_
↪PRESSED);
        lv_obj_add_flag(img, LV_OBJ_FLAG_CLICKABLE);
        lv_obj_add_event_cb(img, event_cb, LV_EVENT_PRESSED, snapshot_obj);
        lv_obj_add_event_cb(img, event_cb, LV_EVENT_RELEASED, snapshot_
↪obj);
    }
}

#endif
```

9.11.3 API

lv_snapshot.h

9.12 System Monitor (sysmon)

The System Monitor module provides real-time monitoring of system performance metrics directly on your display. It supports both performance monitoring (CPU usage and FPS) and memory monitoring (used memory and fragmentation).

9.12.1 Dipendenze

- Requires `LV_USE_LABEL = 1` in lv_conf.h
- Requires `LV_USE_OBSERVER = 1` in lv_conf.h
- Requires `LV_USE_SYSMON = 1` in lv_conf.h

9.12.2 Utilizzo

Configurazione

Enable in `lv_conf.h`:

```
/* Main sysmon enable */
#define LV_USE_SYSMON 1
```

(continues on next page)

(continua dalla pagina precedente)

```
/* Performance monitor (CPU% and FPS) */
#define LV_USE_PERF_MONITOR 1

/* Memory monitor (used + fragmentation) */
#define LV_USE_MEM_MONITOR 1

/* Optional: refresh period in ms */
#define LV_SYSMON_REFR_PERIOD_DEF 300

/* Optional: log to console instead of screen */
#define LV_USE_PERF_MONITOR_LOG_MODE 0
```

Creating Monitors

```
/* Create generic monitor */
lv_obj_t * sysmon = lv_sysmon_create(lv_display_get_default());

/* Create performance monitor */
lv_sysmon_show_performance(NULL);  /* NULL = default display */

/* Create memory monitor */
lv_sysmon_show_memory(NULL);
```

Performance Monitor

Tracks: - FPS (Frames Per Second) - CPU usage (%) - Render time (ms) - Flush time (ms) - Self CPU usage (%) if enabled

Display format: .. code-block:

```
32 FPS, 45% CPU
8 ms
```

Where: - Line 1: FPS, Total CPU% - Line 2: Total time (Render | Flush)

9.12.3 Pause and Resume

`lv_sysmon_performance_pause(disp)` pauses the perf monitor.

`lv_sysmon_performance_resume(disp)` resumes the perf monitor.

Memory Monitor

Displays: - Current memory usage (kB and %) - Peak memory usage (kB) - Fragmentation (%)

Display format: .. code-block:

```
24.8 kB (76%)
32.4 kB max, 18% frag.
```

Positioning

Configure positions in lv_conf.h:

```
/* Top-right corner */
#define LV_USE_PERF_MONITOR_POS LV_ALIGN_TOP_RIGHT
```

(continues on next page)

(continua dalla pagina precedente)

```
/* Bottom-right corner */
#define LV_USE_MEM_MONITOR_POS LV_ALIGN_BOTTOM_RIGHT
```

9.12.4 Implementation Details

Initialization

Maintains: - Global memory monitor (`sysmon_mem`) - Per-display performance structures

Performance Measurement

Event-based collection:

Event	Measurement
LV_EVENT_REFR_START	Refresh interval start
LV_EVENT_REFR_READY	Record refresh duration
LV_EVENT_RENDER_START	Render time start
LV_EVENT_RENDER_READY	Record render duration
LV_EVENT_FLUSH_*	Measure flush operations

Timers

- Performance: `perf_update_timer_cb`
- Memory: `mem_update_timer_cb`
- Default period: 300ms (`LV_SYSMON_REFR_PERIOD_DEF`)

9.13 UI Testing

9.13.1 Panoramica

The Test module provides functions to emulate clicks, key presses, encoder turns, time passing, and compare the UI with reference images.

These functions can be easily used in any test framework (such as Unity, GoogleTest, etc.), and assertions can be performed to check if, for example:

- A widget's value is different from the expected value after emulating user inputs.
- The values are incorrect after some time has passed.
- The screen's content is different from the reference image.
- Some event functions are not triggered.
- Ecc.

Note that it is assumed the tests are performed on a desktop or server environment, where there are no memory constraints.

9.13.2 Utilizzo

The Test module can be enabled by configuring `LV_USE_TEST` to a non-zero value, and it consists of the following components:

- Helpers
- Display emulation
- Input device emulation

- Screenshot comparison

Helpers

Time

To emulate elapsed time, two functions can be used:

1. `lv_test_wait(ms)`: Emulates that `ms` milliseconds have elapsed, but it also calls `lv_timer_handler` after each millisecond. This is useful to check if events (e.g., long press, long press repeat) and timers were triggered correctly over time.
2. `lv_test_fast_forward(ms)`: Jumps `ms` milliseconds ahead and calls `lv_timer_handler` only once at the end.

`lv_refr_now(NULL)` is called at the end of both functions to ensure that animations and widget coordinates are recalculated.

`lv_refr_now(NULL)` can also be called manually to force LVGL to refresh the emulated display.

Memory Usage

If `LV_USE_STDLIB_MALLOC` is set to `LV_STDLIB_BUILTIN`, memory usage and memory leaks can be monitored.

```
size_t mem1 = lv_test_get_free_mem();
<create and delete items>
size_t mem2 = lv_test_get_free_mem();
if(mem1 != mem2) fail();
```

It might make sense to create and delete items in a loop many times and add a small tolerance to the memory leakage test. This might be needed due to potential memory fragmentation. Empirically, a tolerance of 32 bytes is recommended.

```
if(LV_ABS((int64_t)mem2 - (int64_t)mem1) > 32) fail();
```

Display Emulation

By calling `lv_test_display_create(hor_res, ver_res)`, a dummy display can be created.

It functions like any other normal display, but its content exists only in memory.

When creating this display, the horizontal and vertical resolutions must be passed. Internally, a framebuffer will be allocated for this size, and `XRGB8888` color format will be set.

The resolution and color format can be changed at any time by calling `lv_display_set_resolution()` and `lv_display_set_color_format()`.

Input Device Emulation

By calling `lv_test_indev_create_all()`, three test input devices will be created:

1. A pointer (for touch or mouse)
2. A keypad
3. An encoder

For example, this is how a scroll gesture can be emulated:

```
lv_test_mouse_move_to(20, 30);
lv_test_mouse_press();
lv_test_wait(20);
lv_test_mouse_move_by(0, 100);
```

(continues on next page)

(continua dalla pagina precedente)

```
lv_test_wait(20);
lv_test_mouse_release();
lv_test_wait(20);
```

It is recommended to add *lv_test_wait()* after user actions to ensure that the new state and coordinates are read and applied from the input device.

After that, the user can check if the given widget was really scrolled by getting the Y coordinate of a child.

```
int32_t y_start = lv_obj_get_y(child);
<scroll emulation>
int32_t y_end = lv_obj_get_y(child);
if(y_start + 100 != y_end) fail();
```

Please refer to *lv_test_indev.h* for the list of supported input device emulation functions.

Screenshot Comparison

`bool lv_test_screenshot_compare(const char * fn_ref)` is a useful function to compare the content of the emulated display with reference PNG images.

This function works in a practical way:

- If the folder(s) referenced in `fn_ref` do not exist, they will be created automatically.
- If the reference image is not found, it will be created automatically from the rendered screen.
- If the comparison fails, an `<image_name>_err.png` file will be created with the rendered content next to the reference image.
- If the comparison fails, the X and Y coordinates of the first divergent pixel, along with the actual and expected colors, will also be printed.

The reference PNG images should have a **32-bit color format** and match the display size.

The test display's content will be converted to `XRGB8888` to simplify comparison with the reference images. The conversion is supported from the following formats (i.e., the test display should have a color format in this list):

- *LV_COLOR_FORMAT_XRGB8888*
- *LV_COLOR_FORMAT_ARGB8888*
- *LV_COLOR_FORMAT_ARGB8888_PREMULTIPLIED*
- *LV_COLOR_FORMAT_RGB888*
- *LV_COLOR_FORMAT_RGB565*
- *LV_COLOR_FORMAT_RGB565_SWAPPED*
- *LV_COLOR_FORMAT_L8*
- *LV_COLOR_FORMAT_AL88*
- *LV_COLOR_FORMAT_I1*

To read and decode PNG images and to store the converted rendered image, a few MBs of RAM are dynamically allocated using the standard `malloc` (not *lv_malloc*).

The screenshot comparison uses `lodepng` which is built-in to LVGL and just needs to be ebnabled with `LV_USE_LODEPNG`.

To avoid making the entire Test module dependent on `lodepng`, screenshot comparison can be individually enabled by `LV_USE_TEST_SCREENSHOT_COMPARE`.

9.13.3 API

lv_test_display.h

lv_test_helpers.h

lv_test_indev.h

lv_test_indev_gesture.h

lv_test_private.h

lv_test_screenshot_compare.h

9.14 Translation

9.14.1 Panoramica

LVGL supports two ways of handling translations:

- lv_i18n: A comprehensive tool that extracts translatable strings from C files into YAML files, and generates C translation files from them. It also supports plural forms. See its README for details.
- `lv_translation`: A simpler yet more flexible solution that allows adding translations statically or dynamically. This is the method documented here.

9.14.2 Add Translations

Static Translations

If most translations are known at compile time, they can be defined using string arrays:

```
static const char * languages[] = {"en", "de", "es", NULL};
static const char * tags[] = {"tiger", "lion", "rabbit", "elephant", NULL};
static const char * translations[] = {
    "The Tiger", "Der Tiger", "El Tigre",
    "The Lion", "Der Löwe", "El León",
    "The Rabbit", "Das Kaninchen", "El Conejo",
    "The Elephant", "Der Elefant", "El Elefante",
};

lv_translation_add_static(languages, tags, translations);
```

This method uses only a little extra RAM, as only the pointers to the strings are stored.

Dynamic Translations

If translations are only available at runtime (e.g., from files, serial ports, or online sources), they can be added dynamically.

This approach involves memory allocation. See the example at the bottom of this page for reference.

9.14.3 Select a Language

Once translations are registered, use:

`lv_translation_set_language``("language")`

to set the current language. The parameter must match one of the language names provided during registration.

9.14.4 Translate Strings

To retrieve a translation for a given tag, use:

- `lv_translation_get("tag")`
- or the shorthand: `lv_tr("tag")`

These return a translated string which can be used with widgets:

```
lv_label_set_text(label, lv_tr("settings"));
lv_dropdown_set_options(dd, lv_tr("color_list"));
```

Fallbacks

If a tag exists but the translation for the selected language is missing the tag itself will be returned.

If the tag is not found at all, the tag itself will be used as a fallback as well.

9.14.5 Esempio

Simple translation example

```
#include "../../lv_examples.h"
#if LV_USE_TRANSLATION && LV_BUILD_EXAMPLES

static void add_static(void)
{
    static const char * languages[] = {"en", "de", "es", NULL};
    static const char * tags[] = {"tiger", "lion", "rabbit", "elephant",␣
↪NULL};
    static const char * translations[] = {
        "The Tiger", "Der Tiger", "El Tigre",
        "The Lion", "Der Löwe", "El León",
        "The Rabbit", "Das Kaninchen", "El Conejo",
        "The Elephant", "Der Elefant", "El Elefante",
    };

    lv_translation_add_static(languages, tags, translations);
}

static void add_dynamic(void)
{
    lv_translation_pack_t * pack = lv_translation_add_dynamic();
    lv_translation_add_language(pack, "en");
    lv_translation_add_language(pack, "de");

    lv_translation_tag_dsc_t * tag;
    tag = lv_translation_add_tag(pack, "table");
    lv_translation_set_tag_translation(pack, tag, 0, "It's a table");
    lv_translation_set_tag_translation(pack, tag, 1, "Das is ein Tish");

    tag = lv_translation_add_tag(pack, "chair");
    lv_translation_set_tag_translation(pack, tag, 0, "It's a chair");
    lv_translation_set_tag_translation(pack, tag, 1, "Das ist ein Stuhl");
}

/**
 * Create and use translations
 */
```

(continues on next page)

(continua dalla pagina precedente)

```
void lv_example_translation_1(void)
{
    add_static();
    add_dynamic();

    lv_translation_set_language("de");

    lv_obj_t * label;

    label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, lv_tr("tiger"));

    label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, lv_tr("chair"));
    lv_obj_set_y(label, 50);
}

#endif /*LV_USE_TRANSLATION && LV_BUILD_EXAMPLES*/
```

9.14.6 API

lv_translation.h

lv_translation_private.h

CAPITOLO 10

3rd-Party Libraries

10.1 Arduino ESP littlefs

LittleFS is a little fail-safe filesystem designed for microcontrollers and integrated in the Arduino framework when used with ESP32 and ESP8266.

For a detailed introduction, see:

- https://github.com/esp8266/Arduino
- https://github.com/espressif/arduino-esp32

10.1.1 Utilizzo

Enable *LV_USE_FS_ARDUINO_ESP_LITTLEFS* and define a `LV_FS_ARDUINO_ESP_LITTLEFS_LETTER` in `lv_conf.h`.

10.1.2 API

10.2 Arduino SD

Enables reading from and writing to SD cards. Once an SD memory card is connected to the SPI interface of the Arduino or Genuino board, you can create files and read from, and write to them. You can also move through directories on the SD card.

For a detailed introduction, see:

- https://www.arduino.cc/reference/en/libraries/sd/

10.2.1 Utilizzo

Enable *LV_USE_FS_ARDUINO_SD* and define a `LV_FS_ARDUINO_SD_LETTER` in `lv_conf.h`. You will need to initialize the SD card before LVGL can use it (i.e. `SD.begin(0, SPI, 40000000)`).

10.2.2 API

10.3 Barcode

The LVGL Barcode utility enables you to generate Code-128 bar codes. It uses the code128 library by fhunleth.

10.3.1 Utilizzo

Set *LV_USE_BARCODE* to `1` in `lv_conf.h`.

Use *lv_barcode_create()* to create a barcode object, and use *lv_barcode_update()* to generate a barcode.

Call *lv_barcode_set_scale()* to adjust scaling, call *lv_barcode_set_dark_color()* and *lv_barcode_set_light_color()* adjust colors. Call *lv_barcode_set_direction()* to set the bar code's orientation.

By default, *LV_BARCODE_ENCODING_CODE128_GS1* encoding is used and strips `[FCN1]` and spaces. Optionally use *lv_barcode_set_encoding()* to set *LV_BARCODE_ENCODING_CODE128_RAW*.

After any of a bar code's settings have changed, call *lv_barcode_update()* again to regenerate it.

10.3.2 Notes

- It is best not to manually set the width of the barcode, because when the width of the Widget is lower than the width of the barcode, the display will be incomplete due to truncation.
- The scale adjustment can only be an integer multiple, for example, *lv_barcode_set_scale*`(barcode, 2)` means 2x scaling.
- The direction setting can be *LV_DIR_HOR* or *LV_DIR_VER*.

10.3.3 Esempio

Create a Barcode

```
#include "../../lv_examples.h"
#if LV_USE_BARCODE && LV_BUILD_EXAMPLES

/**
 * Create a Barcode
 */
void lv_example_barcode_1(void)
{
    lv_color_t bg_color = lv_palette_lighten(LV_PALETTE_LIGHT_BLUE, 5);
    lv_color_t fg_color = lv_palette_darken(LV_PALETTE_BLUE, 4);

    lv_obj_t * barcode = lv_barcode_create(lv_screen_active());
    lv_obj_set_height(barcode, 50);
    lv_obj_center(barcode);

    /*Set color*/
    lv_barcode_set_dark_color(barcode, fg_color);
    lv_barcode_set_light_color(barcode, bg_color);

    /*Add a border with bg_color*/
    lv_obj_set_style_border_color(barcode, bg_color, 0);

    /*Set data*/
    lv_barcode_update(barcode, "https://lvgl.io");
```

(continues on next page)

(continua dalla pagina precedente)

```
}

#endif
```

10.3.4 API

lv_barcode.h

lv_barcode_private.h

10.4 BMP Decoder

This BMP Decoder utility allows you to use images from .BMP files in LVGL.

Library source: https://github.com/caj-johnson/bmp-decoder

Instead of loading the whole image at once, BMP pixels are read on demand, so using BMP images requires very little RAM.

If enabled in `lv_conf.h` by setting *LV_USE_BMP* to `1`, LVGL will register a new image decoder automatically so BMP files can be directly used as image sources. Esempio:

```
lv_image_set_src(my_img, "S:path/to/picture.bmp");
```

Note that, a *File System (lv_fs_drv)* driver needs to registered to open images from files. Follow the instructions in *File System (lv_fs_drv)*.

10.4.1 Limitazioni

- Only BMP files are supported. BMP images as C arrays (`lv_image_dsc_t`) are not. This is because there is no practical differences between how the BMP files and LVGL's image format stores the image data.
- BMP files can be loaded only from .BMP files. If you want to store them in flash it's better to convert them to a C array with LVGL's image converter.
- The BMP file's color format needs to match the configured *LV_COLOR_DEPTH* of the display on which it will be rendered. You can use GIMP to save the image in the required format. Both RGB888 and ARGB888 works with *LV_COLOR_DEPTH* `32`
- Color palettes are not supported.
- Because the whole image is not loaded, it cannot be zoomed or rotated.

10.4.2 Esempio

Open a BMP image from file

```
#include "../../lv_examples.h"
#if LV_USE_BMP && LV_BUILD_EXAMPLES

/**
 * Open a BMP file from a file
 */
void lv_example_bmp_1(void)
{
    lv_obj_t * img = lv_image_create(lv_screen_active());
    /* Assuming a File system is attached to letter 'A'
     * E.g. set LV_USE_FS_STDIO 'A' in lv_conf.h */
    lv_image_set_src(img, "A:lvgl/examples/libs/bmp/example_32bit.bmp");
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_obj_center(img);

}

#endif
```

10.4.3 API

lv_bmp.h

10.5 FFmpeg Support

FFmpeg is a complete, cross-platform solution to record, convert and stream audio and video.

The FFmpeg is an LVGL extension that interfaces with the official FFmpeg library to help you add platform-independent recording, converting and streaming audio and video into your LVGL UI.

The set-up steps below are for Linux, but they can be adapted for other platforms.

For a detailed introduction, see: https://www.ffmpeg.org

10.5.1 Installing FFmpeg

```
sudo apt install libavformat-dev libavcodec-dev libswscale-dev libavutil-
↪dev
```

Or download the FFmpeg library from its official download page, then install it:

```
./configure --disable-all --disable-autodetect --disable-podpages --
↪disable-asm --enable-avcodec --enable-avformat --enable-decoders --
↪enable-encoders --enable-demuxers --enable-parsers --enable-protocol=
↪'file' --enable-swscale --enable-zlib
make
sudo make install
```

10.5.2 Adding FFmpeg to Your Project

To use the `FFmpeg` library in your project, you will need to link against these libraries:

libavformat
: part of FFmpeg library

libavcodec
: part of FFmpeg library

libavutil
: part of FFmpeg library

libswscale
: part of FFmpeg library

libm
libz
libpthread

If you are using GCC-based toolchain, this can be taken care of by adding the following command-line options:

```
-lavformat -lavcodec -lavutil -lswscale -lm -lz -lpthread
```

10.5.3 Utilizzo

Set the *LV_USE_FFMPEG* in `lv_conf.h` to `1`.

Also set `LV_FFMPEG_PLAYER_USE_LV_FS` in `lv_conf.h` to `1` if you want to integrate the LVGL *File System (lv_fs_drv)* extension into FFmpeg. This library can load videos and images. The LVGL file system will always be used when an image is loaded with *lv_image_set_src()* regardless of the value of `LV_FFMPEG_PLAYER_USE_LV_FS`.

See the examples below for how to correctly use this library.

10.5.4 Eventi

- *LV_EVENT_READY* Sent when playback is complete and auto-restart is not enabled.

Learn more about *Eventi*.

10.5.5 Esempi

Decode image

```
#include "../../lv_examples.h"
#if LV_BUILD_EXAMPLES
#if LV_USE_FFMPEG && LV_FFMPEG_PLAYER_USE_LV_FS

/**
 * Open an image from a file
 */
void lv_example_ffmpeg_1(void)
{
    /*It always uses the LVGL filesystem abstraction (not the OS␣
↪filesystem)
     *to open the image, unlike `lv_ffmpeg_player_set_src` which depends on
     *the setting of `LV_FFMPEG_PLAYER_USE_LV_FS`.*/
    lv_obj_t * img = lv_image_create(lv_screen_active());
    lv_image_set_src(img, "A:lvgl/examples/libs/ffmpeg/ffmpeg.png");
    lv_obj_center(img);
}

#else

void lv_example_ffmpeg_1(void)
{
    /*TODO
     *fallback for online examples*/

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "FFmpeg is not installed");
    lv_obj_center(label);
}

#endif
#endif
```

Decode video

```
#include "../../lv_examples.h"
#if LV_BUILD_EXAMPLES
#if LV_USE_FFMPEG
```

(continues on next page)

(continua dalla pagina precedente)

```
#if LV_FFMPEG_PLAYER_USE_LV_FS
    #define PATH_PREFIX "A:"
#else
    #define PATH_PREFIX "./"
#endif

/**
 * Open a video from a file
 */
void lv_example_ffmpeg_2(void)
{
    /*birds.mp4 is downloaded from http://www.videezy.com (Free Stock␣
↪Footage by Videezy!)
     *https://www.videezy.com/abstract/44864-silhouettes-of-birds-over-the-
↪sunset*/
    /*It will use the LVGL filesystem abstraction (not the OS filesystem)
     *if `LV_FFMPEG_PLAYER_USE_LV_FS` is set.*/
    lv_obj_t * player = lv_ffmpeg_player_create(lv_screen_active());
    lv_ffmpeg_player_set_src(player, PATH_PREFIX "lvgl/examples/libs/
↪ffmpeg/birds.mp4");
    lv_ffmpeg_player_set_auto_restart(player, true);
    lv_ffmpeg_player_set_cmd(player, LV_FFMPEG_PLAYER_CMD_START);
    lv_obj_center(player);
}

#else

void lv_example_ffmpeg_2(void)
{
    /*TODO
     *fallback for online examples*/

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "FFmpeg is not installed");
    lv_obj_center(label);
}

#endif
#endif
```

10.5.6 API

lv_ffmpeg.h

lv_ffmpeg_private.h

10.6 FreeType Support

FreeType is a freely available software library to render fonts.

The LVGL FreeType extension is an interface to the FreeType library, enabling you to generate font bitmaps at run time from most vector- and bitmap-font file formats.

For a detailed introduction, see: https://freetype.org/ .

10.6.1 Adding FreeType to Your Project

First, Download FreeType from the `freetype2` folder (and optionally `freetype-docs` and `freetype-demos`) from its official repository. (The latest version is recommended.)

If you haven't already done so, now is a good time to get familiar with setting up and configuring this library. The above website is a good place to start, as is the `README` file in the top directory of the version you downloaded.

There are two ways to use FreeType:

For UNIX

For UNIX-like systems, the following is recommended to compile and install FreeType libraries.

- Enter the FreeType source code directory
- `make`
- `sudo make install`
- Add include path: `/usr/include/freetype2` (for GCC: `-I/usr/include/freetype2 -L/usr/local/lib`)
- Link against library: `freetype` (for GCC: `-L/usr/local/lib -lfreetype`)

For Embedded Devices

For embedded devices, it is recommended to use the FreeType configuration files provided by LVGL:

- lvgl/src/libs/freetype/ftmodule.h
- lvgl/src/libs/freetype/ftoption.h

which only include the most commonly used modules and options, which is important to save limited FLASH space.

- Copy the FreeType source code to your project directory.
- Refer to the following `Makefile` for configuration:

```
# FreeType custom configuration header file
CFLAGS += -DFT2_BUILD_LIBRARY
CFLAGS += -DFT_CONFIG_MODULES_H=<lvgl/src/libs/freetype/ftmodule.h>
CFLAGS += -DFT_CONFIG_OPTIONS_H=<lvgl/src/libs/freetype/ftoption.h>

# FreeType include path
CFLAGS += -Ifreetype/include

# FreeType C source file
FT_CSRCS += freetype/src/base/ftbase.c
FT_CSRCS += freetype/src/base/ftbitmap.c
FT_CSRCS += freetype/src/base/ftdebug.c
FT_CSRCS += freetype/src/base/ftglyph.c
FT_CSRCS += freetype/src/base/ftinit.c
FT_CSRCS += freetype/src/base/ftstroke.c
FT_CSRCS += freetype/src/cache/ftcache.c
FT_CSRCS += freetype/src/gzip/ftgzip.c
FT_CSRCS += freetype/src/sfnt/sfnt.c
FT_CSRCS += freetype/src/smooth/smooth.c
FT_CSRCS += freetype/src/truetype/truetype.c
CSRCS += $(FT_CSRCS)
```

10.6.2 Utilizzo

Set *LV_USE_FREETYPE* in `lv_conf.h` to `1`.

Cache configuration:

- `LV_FREETYPE_CACHE_FT_GLYPH_CNT` Maximum number of cached glyphs., etc.

By default, the FreeType extension doesn't use LVGL's file system. You can simply pass the path to the font as usual on your operating system or platform.

If you want FreeType to use LVGL's memory allocation and file system interface, you can enable `LV_FREETYPE_USE_LVGL_PORT` in `lv_conf.h`, convenient for unified management. If you do this, you will need to exclude the configured FreeType library's `ftsystem.c` file from being compiled, since LVGL's `lv_ftsystem.c` has custom versions of the functions defined therein.

The font style supports *Italic* and **Bold** fonts processed by software, and can be set by using following values where style values are required:

- *LV_FREETYPE_FONT_STYLE_NORMAL*: Default style.
- *LV_FREETYPE_FONT_STYLE_ITALIC*: Italic style.
- *LV_FREETYPE_FONT_STYLE_BOLD*: Bold style.

These values can be combined, e.g. *LV_FREETYPE_FONT_STYLE_BOLD* | *LV_FREETYPE_FONT_STYLE_ITALIC*.

The FreeType extension also supports colored bitmap glyphs such as emojis. Note that only bitmaps are supported at this time. Colored vector graphics cannot be rendered. An example on how to draw a colored bitmap glyph is shown below.

Use the *lv_freetype_font_create()* function to create a font. To delete a font, use *lv_freetype_font_delete()*. For more detailed usage, please refer to the example code below.

Further Reading

- FreeType tutorial
- LVGL's *Adding a New Font*

Rendering vector fonts is supported with VGLite or ThorVG, when using vector fonts with ThorVG, it is possible to set a letter outline of a different color.

This is achieved by setting the style attributes with the `lv_style_set_text_outline_width()` and `lv_style_set_text_outline_color()` functions

You will have to account for the increased width and height of letter due to the added letter outline, to avoid letters overlapping space them out using *lv_style_set_text_letter_space()*

To use vector fonts with ThorVG, you will have to enable `LV_USE_VECTOR_GRAPHICS` in `lv_conf.h`

Nota

This feature is currently experimental, there are clipping issues especially when using large font sizes.

See the `lv_example_freetype_2_vector_font()` function for a usage example

10.6.3 Esempi

Create a font with FreeType

```
#include "../../lv_examples.h"
#if LV_BUILD_EXAMPLES
#if LV_USE_FREETYPE

#if LV_FREETYPE_USE_LVGL_PORT
    #define PATH_PREFIX "A:"
#else
    #define PATH_PREFIX "./"
#endif

/**
 * Load a font with FreeType
 */
void lv_example_freetype_1(void)
{
    /*Create a font*/
    lv_font_t * font = lv_freetype_font_create(PATH_PREFIX "lvgl/examples/
↪libs/freetype/Lato-Regular.ttf",
                                               LV_FREETYPE_FONT_RENDER_
↪MODE_BITMAP,
                                               24,
                                               LV_FREETYPE_FONT_STYLE_
↪NORMAL);

    if(!font) {
        LV_LOG_ERROR("freetype font create failed.");
        return;
    }

    /*Create style with the new font*/
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_text_font(&style, font);
    lv_style_set_text_align(&style, LV_TEXT_ALIGN_CENTER);

    /*Create a label with the new style*/
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_add_style(label, &style, 0);
    lv_label_set_text(label, "Hello world\nI'm a font created with FreeType
↪");
    lv_obj_center(label);
}
#else

void lv_example_freetype_1(void)
{
    /*TODO
     *fallback for online examples*/

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "FreeType is not installed");
    lv_obj_center(label);
}

#endif
#endif
```

Use a bitmap font to draw Emojis using FreeType

```
#include "../../lv_examples.h"
#if LV_BUILD_EXAMPLES
#if LV_USE_FREETYPE

#if LV_FREETYPE_USE_LVGL_PORT
    #define PATH_PREFIX "A:"
#else
    #define PATH_PREFIX "./"
#endif

/*
 * Load a vector font
 * ThorVG needs to be enabled, LV_USE_VECTOR_GRAPHICS=1
 */
void lv_example_freetype_2_vector_font(uint32_t font_size, uint32_t border_
↪width)
{
    lv_font_t * font = lv_freetype_font_create(PATH_PREFIX "lvgl/examples/
↪libs/freetype/Lato-Regular.ttf",
                                               LV_FREETYPE_FONT_RENDER_
↪MODE_OUTLINE,
                                               font_size,
                                               LV_FREETYPE_FONT_STYLE_
↪NORMAL);


    if(!font) {
        LV_LOG_ERROR("Freetype font create failed.");
        return;
    }

    /*Create style with the new font*/
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_text_font(&style, font);
    lv_style_set_text_align(&style, LV_TEXT_ALIGN_CENTER);
    lv_style_set_text_color(&style, lv_color_hex(0xFF0000));
    lv_style_set_text_opa(&style, LV_OPA_100);
    lv_style_set_text_outline_stroke_opa(&style, LV_OPA_100);
    lv_style_set_text_outline_stroke_color(&style, lv_color_hex(0x00FF00));
    lv_style_set_text_outline_stroke_width(&style, border_width);

    /*Avoid overlapping issue when using letter outlines*/
    lv_style_set_text_letter_space(&style, border_width);
    lv_style_set_text_line_space(&style, border_width);

    /*Create a label with the new style*/
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_add_style(label, &style, 0);
    lv_label_set_text(label,
                      "Hello world\nI'm a font created with FreeType\
↪nHello world\nI'm a font created with FreeType\nHello world\nI'm a font␣
↪created with FreeType\nHello world\nI'm a font created with FreeType\
↪nHello world\nI'm a font created with FreeType\nHello world\nI'm a font␣
↪created with FreeType\nHello world\nI'm a font created with FreeType\
```

(continues on next page)

(continua dalla pagina precedente)

```
↪nHello world\nI'm a font created with FreeType\nHello world\nI'm a font␣
↪created with FreeType\nHello world\nI'm a font created with FreeType\
↪nHello world\nI'm a font created with FreeType\nHello world\nI'm a font␣
↪created with FreeType\nHello world\nI'm a font created with FreeType\
↪nHello world\nI'm a font created with FreeType\nHello world\nI'm a font␣
↪created with FreeType\nHello world\nI'm a font created with FreeType\
↪nHello world\nI'm a font created with FreeType\nHello world\nI'm a font␣
↪created with FreeType\nHello world\nI'm a font created with FreeType\
↪nHello world\nI'm a font created with FreeType\nHello world\nI'm a font␣
↪created with FreeType\nHello world\nI'm a font created with FreeType\
↪nHello world\nI'm a font created with FreeType\nHello world\nI'm a font␣
↪created with FreeType\nHello world\nI'm a font created with FreeType\
↪nHello world\nI'm a font created with FreeType\nHello world\nI'm a font␣
↪created with FreeType\nHello world\n"
                      "I'm a font created with FreeType\nHello world\nI'm␣
↪a font created with FreeType\nHello world\nI'm a font created with␣
↪FreeType\nHello world\nI'm a font created with FreeType\n");

}

/**
 * Load a font with FreeType
 */
void lv_example_freetype_2(void)
{
    /*Create a font*/
    lv_font_t * font = lv_freetype_font_create(PATH_PREFIX "lvgl/examples/
↪libs/freetype/Lato-Regular.ttf",
                                               LV_FREETYPE_FONT_RENDER_
↪MODE_BITMAP,
                                               400,
                                               LV_FREETYPE_FONT_STYLE_
↪NORMAL);

    /* this font is created from a downscaled NotoColorEmoji to 34x32px
     * Subset containing only a single emoji was created using fonttools:
     * Command: fonttools subset NotoColorEmoji.ttf --text=□ */
    lv_font_t * font_emoji = lv_freetype_font_create(PATH_PREFIX "lvgl/
↪examples/libs/freetype/NotoColorEmoji-32.subset.ttf",
                                                     LV_FREETYPE_FONT_
↪RENDER_MODE_BITMAP,
                                                     200,
                                                     LV_FREETYPE_FONT_
↪STYLE_NORMAL);

    if(!font || !font_emoji) {
        LV_LOG_ERROR("freetype font create failed.");
        return;
    }

    font->fallback = font_emoji;

    /*Create style with the new font*/
    static lv_style_t style;
    lv_style_init(&style);
    lv_style_set_text_font(&style, font);
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_style_set_text_align(&style, LV_TEXT_ALIGN_CENTER);

    /*Create a label with the new style*/
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_add_style(label, &style, 0);
    lv_label_set_text(label, "Hello world\nI'm a font created with␣
↪FreeType □");
    lv_obj_center(label);
}
#else

void lv_example_freetype_2_vector_font(uint32_t font_size, uint32_t border_
↪width)
{
    /*TODO
     *fallback for online examples*/
    LV_UNUSED(font_size);
    LV_UNUSED(border_width);

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "FreeType is not installed");
    lv_obj_center(label);
}

void lv_example_freetype_2(void)
{
    /*TODO
     *fallback for online examples*/

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "FreeType is not installed");
    lv_obj_center(label);
}

#endif
#endif
```

Freetype font kerning

```
#include "../../lv_examples.h"
#if LV_BUILD_EXAMPLES
#if LV_USE_FREETYPE

#if LV_FREETYPE_USE_LVGL_PORT
    #define PATH_PREFIX "A:"
#else
    #define PATH_PREFIX "./"
#endif

#define KERNING_TEST_TEXT "AVAWAY,ToTaTe"

static void create_label(lv_font_kerning_t kerning, int32_t y_ofs, const␣
↪char * text)
{
    lv_font_info_t font_info;
    lv_freetype_init_font_info(&font_info);
```

(continues on next page)

(continua dalla pagina precedente)

```
    font_info.name = PATH_PREFIX "lvgl/examples/libs/freetype/Lato-Regular.
↪ttf";
    font_info.size = 32;
    font_info.kerning = kerning;

    lv_font_t * font = lv_freetype_font_create_with_info(&font_info);
    if(!font) {
        LV_LOG_ERROR("freetype font create failed.");
        return;
    }

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_set_style_text_font(label, font, 0);
    lv_label_set_text(label, text);
    lv_obj_align(label, LV_ALIGN_CENTER, 0, y_ofs);
}

/**
 * FreeType kerning example
 */
void lv_example_freetype_3(void)
{
    create_label(LV_FONT_KERNING_NONE, -32, "Kerning None: " KERNING_TEST_
↪TEXT);
    create_label(LV_FONT_KERNING_NORMAL, 32, "Kerning Normal: " KERNING_
↪TEST_TEXT);
}
#else

void lv_example_freetype_3(void)
{
    /*TODO
     *fallback for online examples*/

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "FreeType is not installed");
    lv_obj_center(label);
}

#endif
#endif
```

10.6.4 API

lv_freetype.h

lv_freetype_private.h

10.7 File System Interfaces

LVGL's *File System (lv_fs_drv)* module provides an abstraction that enables you to attach any type of file system for LVGL's use. File systems already supported are:

- FATFS
- STDIO (Linux and Windows using C standard function .e.g `fopen`, `fread`)
- POSIX (Linux and Windows using POSIX function .e.g `open`, `read`)

- WIN32 (Windows using Win32 API function .e.g `CreateFileA`, `ReadFile`)
- MEMFS (read a file from a memory buffer)
- LITTLEFS (a little fail-safe filesystem designed for microcontrollers)
- Arduino ESP LITTLEFS (a little fail-safe filesystem designed for Arduino ESP)
- Arduino SD (allows for reading from and writing to SD cards)

You still need to provide the drivers and libraries, this extension provides only the bridge between LVGL and these file systems.

10.7.1 Utilizzo

In `lv_conf.h` enable `LV_USE_FS_...` (by setting its value to `1`) and assign an upper cased letter to `LV_FS_..._DRIVER_LETTER` (e.g. `'S'`). If more than one file system is enabled, each driver will need to have a unique driver-identifier letter. After that you can access files using that driver letter. Example with driver identifier letter `'S'`:

Linux-like relative path
`"S:path/to/file.txt"`

Linux-like absolute path
`"S:/path/to/file.txt"`

Windows-like relative path
`"S:C:path/to/file.txt"`

Windows-like absolute path
`"S:C:/path/to/file.txt"`

Do not confuse the driver-identifier letter with the Windows/DOS/FAT "drive letter", which is part of the path passed to the OS-level functions. For more details, see *Identifier Letters*.

Cached reading is also supported if `LV_FS_..._CACHE_SIZE` is set to a non-zero value.

For further details, including how to create support for your own file system, see *File System (lv_fs_drv)*.

10.8 GIF Decoder

GIF Decoder is an LVGL extension that enables you to use GIF images in your LVGL UI.

For a detailed introduction, see: https://github.com/lecram/gifdec .

10.8.1 Utilizzo

Once enabled in `lv_conf.h` by setting *`LV_USE_GIF`* to `1`, *`lv_gif_create`*`(parent)` can be used to create a gif widget.

`lv_gif_set_src``(widget, src)` works very similarly to *`lv_image_set_src()`*. As source, it also accepts images as variables (`lv_image_dsc_t`) or files.

Converting GIF Files to C Arrays

To convert a GIF file to an array of bytes, use LVGL's online converter. Select "Raw" color format and "C array" Output format.

Using GIF Images from Files

Esempio:

```
lv_gif_set_src(widget, "S:path/to/example.gif");
```

Note that, a file system driver needs to be registered to open images from files. To do so, follow the instructions in *File System (lv_fs_drv)*.

10.8.2 Memory Requirements

To decode and display a GIF animation the following amount of RAM (in bytes) is required for each of the following color depths:

- `LV_COLOR_DEPTH` 8: 3 × image width × image height
- `LV_COLOR_DEPTH` 16: 4 × image width × image height
- `LV_COLOR_DEPTH` 32: 5 × image width × image height

10.8.3 Esempio

Open a GIF image from file and variable

```
#include "../../lv_examples.h"
#if LV_USE_GIF && LV_BUILD_EXAMPLES

/**
 * Open a GIF image from a file and a variable
 */
void lv_example_gif_1(void)
{
    LV_IMAGE_DECLARE(img_bulb_gif);
    lv_obj_t * img;

    img = lv_gif_create(lv_screen_active());
    lv_gif_set_src(img, &img_bulb_gif);
    lv_obj_align(img, LV_ALIGN_LEFT_MID, 20, 0);

    img = lv_gif_create(lv_screen_active());
    /* Assuming a File system is attached to letter 'A'
     * E.g. set LV_USE_FS_STDIO 'A' in lv_conf.h */
    lv_gif_set_src(img, "A:lvgl/examples/libs/gif/bulb.gif");
    lv_obj_align(img, LV_ALIGN_RIGHT_MID, -20, 0);
}

#endif
```

10.8.4 API

lv_gif.h

lv_gif_private.h

10.9 littlefs

littlefs is a little fail-safe filesystem library designed for microcontrollers.

The lv_fs_littlefs extension is an interface to the littlefs library.

For a detailed introduction, see: https://github.com/littlefs-project/littlefs .

10.9.1 Utilizzo

Set *LV_USE_FS_LITTLEFS* in `lv_conf.h` to `1` and define an upper-case letter (as a C character type) for `LV_FS_LITTLEFS_LETTER` in the range ['A'..'Z'].

When enabled *lv_littlefs_set_handler()* can be used to set up a mount point.

10.9.2 Esempio

```
#include "lfs.h"

// configuration of the filesystem is provided by this struct
const struct lfs_config cfg = {
    // block device operations
    .read  = user_provided_block_device_read,
    .prog  = user_provided_block_device_prog,
    .erase = user_provided_block_device_erase,
    .sync  = user_provided_block_device_sync,

    // block device configuration
    .read_size = 16,
    .prog_size = 16,
    .block_size = 4096,
    .block_count = 128,
    .cache_size = 16,
    .lookahead_size = 16,
    .block_cycles = 500,
};

// mount the filesystem
int err = lfs_mount(&lfs, &cfg);

// reformat if we can't mount the filesystem
// this should only happen on the first boot
if (err) {
    lfs_format(&lfs, &cfg);
    lfs_mount(&lfs, &cfg);
}

lv_littlefs_set_handler(&lfs);
```

10.9.3 API

lv_fsdrv.h

See also: lvgl/src/libs/fsdrv/lv_fs_littlefs.c

10.10 libjpeg-turbo Decoder

libjpeg-turbo is an LVGL interface to the libjpeg-turbo library --- a JPEG image codec that uses SIMD instructions to accelerate baseline JPEG compression and decompression on x86, x86-64, Arm, PowerPC, and MIPS systems, as well as progressive JPEG compression on x86, x86-64, and Arm systems.

On such systems, libjpeg-turbo is generally 2-6x as fast as libjpeg, all else being equal.

For a detailed introduction, see: https://libjpeg-turbo.org .

Library source: https://github.com/libjpeg-turbo/libjpeg-turbo

10.10.1 Install

```
sudo apt install libjpeg-turbo8-dev
```

10.10.2 Adding libjpeg-turbo to Your Project

Cmake:

```
find_package(JPEG REQUIRED)
include_directories(${JPEG_INCLUDE_DIR})
target_link_libraries(${PROJECT_NAME} PRIVATE ${JPEG_LIBRARIES})
```

10.10.3 Utilizzo

Set *LV_USE_LIBJPEG_TURBO* in `lv_conf.h` to `1`.

See the examples below.

It should be noted that each image decoded needs to consume:

image width × image height × 3

bytes of RAM, and it needs to be combined with the *Image caching* feature to ensure that the memory usage is within a reasonable range.

10.10.4 Esempio

Load a JPG image

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

#if LV_USE_LIBJPEG_TURBO

/**
 * Load a JPG image
 */
void lv_example_libjpeg_turbo_1(void)
{
    lv_obj_t * wp;

    wp = lv_image_create(lv_screen_active());
    /* Assuming a File system is attached to letter 'A'
     * E.g. set LV_USE_FS_STDIO 'A' in lv_conf.h */
    lv_image_set_src(wp, "A:lvgl/examples/libs/libjpeg_turbo/flower.jpg");
    lv_obj_center(wp);
}

#else

void lv_example_libjpeg_turbo_1(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "LibJPEG-Turbo is not installed");
    lv_obj_center(label);
}

#endif
```

(continues on next page)

(continua dalla pagina precedente)

```
#endif
```

10.10.5 API

lv_libjpeg_turbo.h

10.11 libpng Decoder

libpng is an LVGL interface to the the official PNG reference library, which supports almost all PNG features, is extensible, and has been extensively tested for over 28 years.

For a detailed introduction, see: http://www.libpng.org/pub/png/libpng.html .

10.11.1 Install

```
sudo apt install libpng-dev
```

10.11.2 Adding libpng to Your Project

Cmake:

```
find_package(PNG REQUIRED)
include_directories(${PNG_INCLUDE_DIR})
target_link_libraries(${PROJECT_NAME} PRIVATE ${PNG_LIBRARIES})
```

10.11.3 Utilizzo

Set *LV_USE_LIBPNG* in `lv_conf.h` to `1`.

See the examples below.

It should be noted that each image of this decoder needs to consume

$$\text{width} \times \text{height} \times 4$$

bytes of RAM, and it needs to be combined with the *Image caching* feature to ensure that the memory usage is within a reasonable range. The decoded image is stored in RGBA pixel format.

10.11.4 Esempio

Open a PNG image from file and variable

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

#if LV_USE_LIBPNG

/**
 * Open a PNG image from a file
 */
void lv_example_libpng_1(void)
{
    LV_IMAGE_DECLARE(img_png_demo);
    lv_obj_t * img;

    img = lv_image_create(lv_screen_active());
```

(continues on next page)

(continua dalla pagina precedente)

```
    lv_image_set_src(img, &img_png_demo);
    lv_obj_align(img, LV_ALIGN_LEFT_MID, 10, 0);

    img = lv_image_create(lv_screen_active());
    /* Assuming a File system is attached to letter 'A'
     * E.g. set LV_USE_FS_STDIO 'A' in lv_conf.h */
    lv_image_set_src(img, "A:lvgl/examples/libs/libpng/png_demo.png");
    lv_obj_align(img, LV_ALIGN_RIGHT_MID, -10, 0);
}

#else

void lv_example_libpng_1(void)
{
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "LibPNG is not installed");
    lv_obj_center(label);
}

#endif

#endif
```

10.11.5 API

lv_libpng.h

10.12 LodePNG Decoder

LodePNG is an LVGL interface to the LodePNG library --- a PNG encoder and decoder in C and C++, without any dependencies, giving you an alternate way to use PNG images in your LVGL UI.

For a detailed introduction, see: https://github.com/lvandeve/lodepng .

If enabled in `lv_conf.h` by setting *`LV_USE_LODEPNG`* to `1`, LVGL will register a new image decoder automatically so PNG files can be used directly as an image source.

> **Nota**
>
> A file system driver needs to be registered to open images from files. To do so, follow the instructions in *File System (lv_fs_drv)*.

The whole PNG image is decoded, so

$$\text{width} \times \text{height} \times 4$$

bytes of RAM is required from the LVGL heap. The decoded image is stored in RGBA pixel format.

Since it might take significant time to decode PNG images LVGL's *Image caching* feature can be useful.

10.12.1 Compressing PNG Files

PNG file format supports True color (24/32 bit), and 8-bit palette colors. Usually cliparts, drawings, icons and simple graphics are stored in PNG format, that do not use the whole color space, so it is possible to compress further the image by using 8-bit palette colors, instead of 24/32 bit True color format. Because embedded devices have limited (flash) storage, it is recommended to compress images.

One option is to use a free online PNG compressor site, for example Compress PNG: https://compresspng.com/

10.12.2 Esempio

Open a PNG image from file and variable

```
#include "../../lv_examples.h"
#if LV_USE_LODEPNG && LV_USE_IMAGE && LV_BUILD_EXAMPLES

/**
 * Open a PNG image from a file and a variable
 */
void lv_example_lodepng_1(void)
{
    LV_IMAGE_DECLARE(img_wink_png);
    lv_obj_t * img;

    img = lv_image_create(lv_screen_active());
    lv_image_set_src(img, &img_wink_png);
    lv_obj_align(img, LV_ALIGN_LEFT_MID, 20, 0);

    img = lv_image_create(lv_screen_active());
    /* Assuming a File system is attached to letter 'A'
     * E.g. set LV_USE_FS_STDIO 'A' in lv_conf.h */
    lv_image_set_src(img, "A:lvgl/examples/libs/lodepng/wink.png");
    lv_obj_align(img, LV_ALIGN_RIGHT_MID, -20, 0);
}

#endif
```

10.12.3 API

lv_lodepng.h

10.13 QR Code

The QR-Code-generator library by nayuki is a 3rd-party library that generates QR-Code bitmaps.

The lv_qrcode LVGL extension is an interface to that library which implements a custom Widget that generates and displays QR Codes using the library.

10.13.1 Utilizzo

Enable `LV_USE_QRCODE` in `lv_conf.h` by setting its value to `1`.

Use `lv_qrcode_create()` to create the QR-Code Widget. Then use `lv_qrcode_update()` to generate the QR Code on it.

If you need to re-modify the size and color, use `lv_qrcode_set_size()` and `lv_qrcode_set_dark_color()` or `lv_qrcode_set_light_color()` respectively, and then call `lv_qrcode_update()` again to update the QR Code.

10.13.2 Notes

- QR Codes with less data are smaller, but they are scaled by an integer value to best fit to the given size.

10.13.3 Esempio

Create a QR Code

```
#include "../../lv_examples.h"
#if LV_USE_QRCODE && LV_BUILD_EXAMPLES
#include <string.h>

/**
 * Create a QR Code
 */
void lv_example_qrcode_1(void)
{
    lv_color_t bg_color = lv_palette_lighten(LV_PALETTE_LIGHT_BLUE, 5);
    lv_color_t fg_color = lv_palette_darken(LV_PALETTE_BLUE, 4);

    lv_obj_t * qr = lv_qrcode_create(lv_screen_active());
    lv_qrcode_set_size(qr, 150);
    lv_qrcode_set_dark_color(qr, fg_color);
    lv_qrcode_set_light_color(qr, bg_color);

    /*Set data*/
    const char * data = "https://lvgl.io";
    lv_qrcode_update(qr, data, strlen(data));
    lv_obj_center(qr);

    /*Add a border with bg_color*/
    lv_obj_set_style_border_color(qr, bg_color, 0);
    lv_obj_set_style_border_width(qr, 5, 0);
}

#endif
```

10.13.4 API

lv_qrcode.h

lv_qrcode_private.h

10.14 RLE Decompression

LVGL provides a custom RLE (Run-Length Encoding) decompression method. It can be used to reduce binary image size. The RLE compression is a lossless compression method.

The LVGL's built-in binary image decoder supports RLE-compressed images. The decoder supports both variable and file as image sources. The original binary data is directly decoded to RAM.

10.14.1 Benefits

Most screenshot and UI images (where there are a limited number of colors) can be compressed to save more than 70% space. The below statistics are from a watch project. It shows the file count of every compress level. For rare conditions, RLE compress may increase the file size if there's no large repetition in data.

10.14.2 Theory

The RLE algorithm is a simple compression algorithm that is based on the fact that for many adjacent pixels, the color is the same. The algorithm simply counts how many repeated pixels with the same color there are, and stores the count value and the color value. If the up-coming pixels are not repeated, it stores the non-repeat count value and original color values. For more details, the script used to compress the image can be found from `lvgl/script/LVGLImage.py`.

```
def rle_compress(self, data: bytearray, blksize: int, threshold=16):
    index = 0
    data_len = len(data)
    compressed_data = []
    while index < data_len:
        memview = memoryview(data)
        repeat_cnt = self.get_repeat_count(
            memview[index:], blksize)
        if repeat_cnt == 0:
            # done
            break
        elif repeat_cnt < threshold:
            nonrepeat_cnt = self.get_nonrepeat_count(
                memview[index:], blksize, threshold)
            ctrl_byte = uint8_t(nonrepeat_cnt | 0x80)
            compressed_data.append(ctrl_byte)
            compressed_data.append(
                memview[index: index + nonrepeat_cnt * blksize])
            index += nonrepeat_cnt * blksize
        else:
            ctrl_byte = uint8_t(repeat_cnt)
            compressed_data.append(ctrl_byte)
            compressed_data.append(memview[index: index + blksize])
            index += repeat_cnt * blksize

    return b"".join(compressed_data)
```

10.14.3 Utilizzo

To use the RLE Decoder, enable it in `lv_conf.h` configuration file by setting *LV_USE_RLE* to `1`. The RLE image can then be used in the same way as other images.

```
lv_image_set_src(img, "path/to/image.rle");
```

10.14.4 Generating RLE Compressed Binary Images

An RLE image binary can be directly generated from another image using script `lvgl/script/LVGLImage.py`.

```
./scripts/LVGLImage.py --ofmt BIN --cf I8 --compress RLE cogwheel.png
```

This will decompress `cogwheel.png`, and then re-compress it using RLE and write the output to `./output/cogwheel.bin`.

10.14.5 API

lv_rle.h

10.15 Rlottie Player

Avvertimento

Rlottie is deprecated. Consider using the *Lottie (lv_lottie)* Widget instead.

The Rlottie animation player for LVGL is a 3rd-party extension for LVGL that allows playing Lottie animations in LVGL. It provides an interface to Samsung/rlottie library's C API. This Lottie player is not part of LVGL; it needs to be built separately.

10.15.1 Building Rlottie

To build Samsung's Rlottie, you will need a C++14-compatible compiler and optionally CMake 3.14 or higher.

To build on a desktop you can follow the instructions from Rlottie's README.

In the most basic case it looks like this:

```
mkdir rlottie_workdir
cd rlottie_workdir
git clone https://github.com/Samsung/rlottie.git
mkdir build
cd build
cmake ../rlottie
make -j
sudo make install
```

And finally add the `-lrlottie` flag to your linker.

On embedded systems you need to take care of integrating Rlottie to the given build system.

See the ESP-IDF example below.

10.15.2 Utilizzo

You can use animation from files or raw data (text). In either case first you need to enable *LV_USE_RLOTTIE* in `lv_conf.h` by setting its value to `1`.

The `width` and `height` of the Widget is set in the `lv_rlottie_create_from_...()` function, and the animation will be scaled accordingly.

Use Rlottie from File

To create a Lottie animation from a file, use:

```
lv_obj_t * lottie = lv_rlottie_create_from_file(parent, width, height,
↪"path/to/lottie.json");
```

Note that, Rlottie uses the standard STDIO C file API, so you can use the path "normally" and no LVGL specific driver letter is required.

Use Rlottie from Raw String Data

`lv_example_rlottie_approve.c` contains an example animation in raw format. Instead of storing the JSON string, a hex array is stored for the following reasons:

- avoid escaping " character in the JSON file
- some compilers don't support very long strings

`lvgl/scripts/filetohex.py` can be used to convert a Lottie file a hex array. E.g.:

```
./filetohex.py path/to/lottie.json --filter-character --null-terminate >␣
↪out.txt
```

`--filter-character` filters out non-ASCII characters and `--null-terminate` makes sure that a trailing zero is appended to properly terminate the string.

To create an animation from raw data:

```
extern const uint8_t lottie_data[];
lv_obj_t* lottie = lv_rlottie_create_from_raw(parent, width, height,␣
↪(const char *)lottie_data);
```

10.15.3 Getting Animations

Lottie is standard and popular format so you can find many animation files on the web. For example: https://lottiefiles.com/ .

You can also create your own animations with Adobe After Effects or similar software.

10.15.4 Controlling Animations

LVGL provides two functions to control the animation mode: *lv_rlottie_set_play_mode()* and *lv_rlottie_set_current_frame()*. You'll combine your intentions when calling the first method, like in these examples:

```
lv_obj_t * lottie = lv_rlottie_create_from_file(scr, 128, 128, "test.json
↪");
lv_obj_center(lottie);
// Pause to a specific frame
lv_rlottie_set_current_frame(lottie, 50);
lv_rlottie_set_play_mode(lottie, LV_RLOTTIE_CTRL_PAUSE); // The specified␣
↪frame will be displayed and then the animation will pause
```

(continues on next page)

(continua dalla pagina precedente)

```
// Play backward and loop
lv_rlottie_set_play_mode(lottie, LV_RLOTTIE_CTRL_PLAY | LV_RLOTTIE_CTRL_
↪BACKWARD | LV_RLOTTIE_CTRL_LOOP);

// Play forward once (no looping)
lv_rlottie_set_play_mode(lottie, LV_RLOTTIE_CTRL_PLAY | LV_RLOTTIE_CTRL_
↪FORWARD);
```

The default animation mode is **play forward with loop**.

If you don't enable looping, a *LV_EVENT_READY* is sent when the animation cannot make more progress without looping.

To get the number of frames in an animation or the current frame index, you can cast the `lv_obj_t` instance to a `lv_rlottie_t` instance and inspect the `current_frame` and `total_frames` members.

10.15.5 ESP-IDF Example

Background

Rlottie can be expensive to render on embedded hardware. Lottie animations tend to use a large amount of CPU time and can use large portions of RAM. This will vary from lottie to lottie but in general for best performance:

- Limit total # of frames in the animation
- Where possible, try to avoid bezier type animations
- Limit animation render size

If your ESP32 chip does not have SPIRAM you will face severe limitations in render size.

To give a better idea on this, lets assume you want to render a 240x320 lottie animation.

In order to pass initialization of the lv_rlottie_t object, you need 240x320x32/8 (307k) available memory. The latest ESP32-S3 has 256kb RAM available for this (before FreeRTOS and any other initialization starts taking chunks out). So while you can probably start to render a 50x50 animation without SPIRAM, PSRAM is highly recommended.

Additionally, while you might be able to pass initialization of the `lv_rlottie_t` object, as rlottie renders frame to frame, this consumes additional memory. A 30-frame animation that plays over 1 second probably has minimal issues, but a 300 frame animation playing over 10 seconds could very easily crash due to lack of memory as rlottie renders, depending on the complexity of the animation.

Rlottie will not compile for the IDF using the `-O2` compiler option at this time.

For stability in lottie animations, this author has found that they run best in the IDF when enabling `LV_MEM_CUSTOM` (using `stdlib.h`)

For all its faults, when running right-sized animations, they provide a wonderful utility to LVGL on embedded LCDs and can look really good when done properly.

When picking/designing a lottie animation consider the following limitations:

- Build the lottie animation to be sized for the intended size.
- It can scale/resize, but performance will be best when the base lottie size is as intended.
- Limit total number of frames, the longer the lottie animation is, the more memory it will consume for rendering (rlottie consumes IRAM for rendering).
- Build the lottie animation for the intended frame rate.
- Default lottie is 60fps, embedded LCDs likely won't go above 30fps.

IDF Setup

Where the LVGL simulator uses the installed rlottie lib, the IDF works best when using rlottie as a submodule under the components directory.

```
cd 'your/project/directory'
git add submodule
git add submodule https://github.com/Samsung/rlottie.git ./components/
↪rlottie/rlottie
git submodule update --init --recursive
```

Now, Rlottie is available as a component in the IDF, but it requires some additional changes and a CMakeLists file to tell the IDF how to compile.

Rlottie Patch File

Rlottie relies on dynamic linking for an image loader lib. This needs to be disabled as the IDF doesn't play nice with dynamic linking.

A patch file is available in LVGL under: `/env_support/esp/rlottie/0001-changes-to-compile-with-esp-idf.patch`

Apply the patch file to your rlottie submodule.

CMakeLists for IDF

An example CMakeLists file has been provided at `/env_support/esp/rlottie/CMakeLists.txt`

Copy this CMakeLists file to `'your-project-directory'/components/rlottie/`

In addition to the component CMakeLists file, you'll also need to tell your project level CMakeLists in your IDF project to require rlottie:

```
REQUIRES "lvgl" "rlottie"
```

From here, you should be able to use lv_rlottie objects in your ESP-IDF project as any other widget in LVGL ESP examples. Please remember that these animations can be highly resource constrained and this does not guarantee that every animation will work.

Additional Rlottie Considerations in ESP-IDF

While unnecessary, removing the `rlottie/rlottie/example` folder can remove many un-needed files for this embedded LVGL application.

From here, you can use the relevant LVGL `lv_rlottie...()` functions to create lottie animations in LVGL on embedded hardware!

Please note, that while lottie animations are capable of running on many ESP chips, below is recommended for best performance.

- ESP32-S3-WROOM-1-N16R8
  - 16mb quad spi flash
  - 8mb octal spi PSRAM
- IDF4.4 or higher

The Esp-box devkit meets this spec and https://github.com/espressif/esp-box is a great starting point to adding lottie animations.

You will need to enable *`LV_USE_RLOTTIE`* through **idf.py** menuconfig under LVGL component settings.

Additional Changes to Make Use of SPIRAM

`lv_alloc / realloc` does not make use of SPIRAM. Given the high memory usage of lottie animations, it is recommended to shift as much out of internal DRAM into SPIRAM as possible. In order to do so, SPIRAM will need to be enabled in the menuconfig options for your given espressif chip.

There may be a better solution for this, but for the moment the recommendation is to make local modifications to the LVGL component in your espressif project. This is as simple as swapping `lv_alloc / lv_realloc` calls in `lv_rlottie.c` with `heap_caps_malloc` (for IDF) with the appropriate `MALLOC_CAP` call --- for SPIRAM usage this is `MALLOC_CAP_SPIRAM`.

```
rlottie->allocated_buf = heap_caps_malloc(allocated_buf_size+1, MALLOC_CAP_
↪SPIRAM);
```

10.15.6 Esempio

Load a Lottie animation from raw data

```
#include "../../lv_examples.h"
#if LV_BUILD_EXAMPLES
#if LV_USE_RLOTTIE

/**
 * Load an lottie animation from flash
 */
void lv_example_rlottie_1(void)
{
    extern const uint8_t lv_example_rlottie_approve[];
    lv_obj_t * lottie = lv_rlottie_create_from_raw(lv_screen_active(), 100,
↪ 100, (const char *)lv_example_rlottie_approve);
    lv_obj_center(lottie);
}

#else
void lv_example_rlottie_1(void)
{
    /*TODO
     *fallback for online examples*/

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Rlottie is not installed");
    lv_obj_center(label);
}

#endif
#endif
```

Load a Lottie animation from a file

```
#include "../../lv_examples.h"
#if LV_BUILD_EXAMPLES
#if LV_USE_RLOTTIE

/**
 * Load an lottie animation from file
 */
void lv_example_rlottie_2(void)
{
```

(continues on next page)

(continua dalla pagina precedente)

```
    /*The rlottie library uses STDIO file API, so there is no driver␣
↪letter for LVGL*/
    lv_obj_t * lottie = lv_rlottie_create_from_file(lv_screen_active(),␣
↪100, 100,
                                                    "lvgl/examples/libs/
↪rlottie/lv_example_rlottie_approve.json");
    lv_obj_center(lottie);
}

#else
void lv_example_rlottie_2(void)
{
    /*TODO
     *fallback for online examples*/

    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_label_set_text(label, "Rlottie is not installed");
    lv_obj_center(label);
}

#endif
#endif
```

10.15.7 API

lv_rlottie.h

lv_rlottie_private.h

10.16 SVG Support

The lv_svg extension provides makes it possible to use SVG images in your LVGL UI using the Scalable Vector Graphics (SVG) Tiny 1.2 Specification.

For a detailed introduction, see: https://www.w3.org/TR/SVGTiny12/

10.16.1 Utilizzo

Enable *LV_USE_SVG* in `lv_conf.h` by setting its value to `1`.

See the examples below.

If you need support for SVG animation attribute parsing, you can set *LV_USE_SVG_ANIMATION* in `lv_conf.h` to `1`.

10.16.2 Esempio

```
lv_svg_node_t * svg_doc;
const char* svg_data = "<svg><rect x=\"0\" y=\"0\" width=\"100\" height=\
↪"100\"/></svg>";

/* Create an SVG DOM tree */
svg_doc = lv_svg_load_data(svg_data, svg_len);
...

/* Draw SVG image */
lv_draw_svg(layer, svg_doc);
```

(continues on next page)

(continua dalla pagina precedente)

```
...

/* Release the DOM tree */
lv_svg_node_delete(svg_doc);
```

`lv_image` also supports SVG images, For example:

```
lv_image_set_src(widget, "S:path/to/example.svg");
```

10.16.3 API

lv_svg.h

lv_svg_decoder.h

lv_svg_parser.h

lv_svg_render.h

lv_svg_token.h

10.17 Tiny TTF font engine

The lv_tiny_ttf extension allows using TrueType fonts in LVGL using the stb_truetype 3rd-Party Library.

For a detailed introduction, see: https://github.com/nothings/stb.

10.17.1 Utilizzo

When enabled in `lv_conf.h` by setting *`LV_USE_TINY_TTF`* to `1`, *`lv_tiny_ttf_create_data`*`(data, data_size, font_size)` can be used to create a TTF font instance with the specified line height. You can then use that font anywhere `lv_font_t` is accepted.

By default, the TTF or OTF file must be embedded as an array, either in a header, or loaded into RAM in order to function.

However, if `LV_TINY_TTF_FILE_SUPPORT` is enabled (i.e. `1`), `lv_tiny_ttf_create_file(path, font_size)` will also be available, allowing tiny_ttf to stream from a file. The file must remain open the entire time the font is being used.

After a font is created, you can change the font size in pixels by using *`lv_tiny_ttf_set_size`*`(font, font_size)`.

By default, a font will cache data for up to 256 glyph elements to speed up rendering. This maximum can be changed by using *`lv_tiny_ttf_create_data_ex`*`(data, data_size, font_size, kerning, cache_size)` or `lv_tiny_ttf_create_file_ex(path, font_size, kerning, cache_size)` (when available). The cache size is indicated in number of entries. The `kerning` argument will be one of the `LV_FONT_KERNING_...` values, indicating whether to allow kerning, if supported, or disable.

10.17.2 Esempio

Open a font with Tiny TTF from data array

```
#include "../../lv_examples.h"
#if LV_USE_TINY_TTF && LV_BUILD_EXAMPLES

/**
 * Load a font with Tiny_TTF
 */
void lv_example_tiny_ttf_1(void)
```

(continues on next page)

(continua dalla pagina precedente)

```
{
    extern const uint8_t ubuntu_font[];
    extern const int ubuntu_font_size;

    /*Create style with the new font*/
    static lv_style_t style;
    lv_style_init(&style);
    lv_font_t * font = lv_tiny_ttf_create_data(ubuntu_font, ubuntu_font_
↪size, 30);
    lv_style_set_text_font(&style, font);
    lv_style_set_text_align(&style, LV_TEXT_ALIGN_CENTER);

    /*Create a label with the new style*/
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_add_style(label, &style, 0);
    lv_label_set_text(label, "Hello world\nI'm a font\ncreated\nwith Tiny␣
↪TTF");
    lv_obj_center(label);
}
#endif
```

Load a font with Tiny_TTF from file

```
#include "../../lv_examples.h"
#if LV_USE_TINY_TTF && LV_TINY_TTF_FILE_SUPPORT && LV_BUILD_EXAMPLES

/**
 * Load a font with Tiny_TTF from file
 */
void lv_example_tiny_ttf_2(void)
{
    /*Create style with the new font*/
    static lv_style_t style;
    lv_style_init(&style);
    lv_font_t * font = lv_tiny_ttf_create_file("A:lvgl/examples/libs/tiny_
↪ttf/Ubuntu-Medium.ttf", 30);
    lv_style_set_text_font(&style, font);
    lv_style_set_text_align(&style, LV_TEXT_ALIGN_CENTER);

    /*Create a label with the new style*/
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_add_style(label, &style, 0);
    lv_label_set_text(label, "Hello world\nI'm a font\ncreated\nwith Tiny␣
↪TTF");
    lv_obj_center(label);
}
#endif
```

Change font size with Tiny_TTF

```
#include "../../lv_examples.h"
#if LV_USE_TINY_TTF && LV_BUILD_EXAMPLES && LV_USE_OBSERVER

static void font_size_observer_cb(lv_observer_t * observer, lv_subject_t *␣
↪subject);
```

(continues on next page)

(continua dalla pagina precedente)

```
static lv_subject_t subject_font;

/**
 * Change font size with Tiny_TTF
 */
void lv_example_tiny_ttf_3(void)
{
    extern const uint8_t ubuntu_font[];
    extern const int ubuntu_font_size;

    lv_subject_init_int(&subject_font, 25);

    /*Create style with the new font*/
    static lv_style_t style;
    lv_style_init(&style);
    lv_font_t * font = lv_tiny_ttf_create_data(ubuntu_font, ubuntu_font_
↪size, 25);
    lv_style_set_text_font(&style, font);
    lv_style_set_text_align(&style, LV_TEXT_ALIGN_CENTER);

    lv_obj_t * slider = lv_slider_create(lv_screen_active());
    lv_obj_center(slider);
    lv_slider_set_range(slider, 5, 50);
    lv_obj_align(slider, LV_ALIGN_BOTTOM_MID, 0, -50);
    lv_slider_bind_value(slider, &subject_font);

    lv_obj_t * slider_label = lv_label_create(lv_screen_active());
    lv_obj_align_to(slider_label, slider, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
    lv_label_bind_text(slider_label, &subject_font, "%d");

    /*Create a label with the new style*/
    lv_obj_t * label = lv_label_create(lv_screen_active());
    lv_obj_add_style(label, &style, 0);
    lv_obj_set_size(label, LV_SIZE_CONTENT, LV_SIZE_CONTENT);
    lv_label_set_text(label, "Hello world!");
    lv_obj_center(label);

    lv_subject_add_observer(&subject_font, font_size_observer_cb, &style);
}

static void font_size_observer_cb(lv_observer_t * observer, lv_subject_t *␣
↪subject)
{
    lv_style_t * style = (lv_style_t *) lv_observer_get_user_
↪data(observer);
    lv_style_value_t v;
    lv_style_get_prop(style, LV_STYLE_TEXT_FONT, &v);
    lv_font_t * font = (lv_font_t *) v.ptr;
    int32_t size = lv_subject_get_int(subject);

    lv_tiny_ttf_set_size(font, size);

    lv_obj_report_style_change(style);
}
#endif
```

10.17.3 API

lv_tiny_ttf.h

10.18 Tiny JPEG Decompressor (TJpgDec)

Tiny JPEG Decompressor is an LVGL interface to the TJpgDec library --- a generic JPEG image decompressor module that highly optimized for small embedded systems. It works with very low memory consumption, so that it can be incorporated into tiny microcontrollers, such as AVR, 8051, PIC, Z80, Cortex-M0, etc..

For a detailed introduction, see: TJpgDec.

10.18.1 Panoramica

Features and restrictions:

- JPEG is decoded in 8x8 tiles.
- Only baseline JPEG files are supported (no progressive JPEG support).
- Read from file and C array are implemented.
- Only the required portions of the JPEG images are decoded, therefore they cannot be zoomed or rotated.

10.18.2 Utilizzo

Set `LV_USE_TJPGD` to `1` in `lv_conf.h`. LVGL will register a new image decoder automatically so JPEG files can be used directly as image sources.

Ad esempio:

```
lv_image_set_src(my_img, "S:path/to/picture.jpg");
```

> **Nota**
>
> A file system driver needs to be registered to open images from files. To do so, follow the instructions in *File System (lv_fs_drv)*.

10.18.3 Converter

Converting JPEG to C array

- Use lvgl online tool https://lvgl.io/tools/imageconverter
- Color format = RAW, output format = C Array

10.18.4 Esempio

Load a JPG image

```
#include "../../lv_examples.h"
#if LV_USE_TJPGD && LV_BUILD_EXAMPLES

/**
 * Load a JPG image
 */
void lv_example_tjpgd_1(void)
{
    lv_obj_t * wp;
```

(continues on next page)

(continua dalla pagina precedente)

```
    wp = lv_image_create(lv_screen_active());
    /* Assuming a File system is attached to letter 'A'
     * E.g. set LV_USE_FS_STDIO 'A' in lv_conf.h */
    lv_image_set_src(wp, "A:test_img_lvgl_logo.jpg");
    lv_obj_center(wp);
}

#endif
```

10.18.5 API

CAPITOLO 11

Debugging

11.1 GDB Plug-In

11.1.1 Debugging LVGL with GDB

To facilitate debugging LVGL with GDB, a GDB plugin is provided. This plugin can be found in the `lvgl/scripts/gdb` directory. The GDB plugin can be used with any target where GDB is available. For example, you can use it to debug a device connected to a PC via JLink, which provides a GDB server. Additionally, if your device crashes and you have a core dump, you can use GDB to analyze the core dump. To load the LVGL GDB plugin within GDB's command line, type the following command:

```
source lvgl/scripts/gdb/gdbinit.py
```

Example of usage:

```
(gdb) source lvgl/scripts/gdb/gdbinit.py

(gdb) dump obj -L 2
obj@0x60700000dd10 (0,0,799,599)
tabview@0x608000204ca0 (0,0,799,599)
    obj@0x607000025da0 (0,0,799,69)
    obj@0x607000025e80 (0,70,799,599)
    obj@0x60700002bd70 (743,543,791,591)
    btn@0x60700002c7f0 (747,547,787,587)
keyboard@0x60d0000f7040 (0,300,799,599)
dropdown-list@0x608000205420 (0,0,129,129)
    label@0x60d0000f7ba0 (22,22,56,39)
(gdb)
```

The plugin provides the following commands.

- `dump obj`: Dump the object tree.
- `info style`: Show the object's style.
- `info draw_unit`: Display all current drawing unit information.

11.1.2 Dump obj tree

`dump obj`: Dump the object tree.

`dump obj -L 2`: Dump the object tree with a depth of 2.

`dump obj -a 0x60700000dd10`: Dump the object tree starting from the specified address.

11.1.3 Show obj's style

This command can dump the object's local style, since style value is a union, it's displayed in all possible formats.

`info style address_of_obj`: Show the object's style.

Esempio:

```
(gdb) info style 0x60700000dd10
  32 = {num = 90, ptr = 0x5a, color = {blue = 90 'Z', green = 0 '\000',␣
↪red = 0 '\000'}}
  158 = {num = 32767, ptr = 0x7fff, color = {blue = 255 '\377', green =␣
↪127 '\177', red = 0 '\000'}}
(gdb) p lv_global->disp_default->act_scr
$4 = (lv_obj_t *) 0x60700000dd10
(gdb) info style $4
  32 = {num = 90, ptr = 0x5a, color = {blue = 90 'Z', green = 0 '\000',␣
↪red = 0 '\000'}}
  158 = {num = 32767, ptr = 0x7fff, color = {blue = 255 '\377', green =␣
↪127 '\177', red = 0 '\000'}}
(gdb)
```

11.1.4 Connect to Debugger

This command provides the ability to connect and debug GDB Python Script using IDE.

Connect to `PyCharm` / `VSCode` / `Eclipse (not supported yet)`

`debugger -t pycharm`

`debugger -t vscode`

`debugger -t eclipse`

How to use it specifically, search `pydevd_pycharm` / `debugpy` for details.

11.2 Logging

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

11.2.1 Log Level

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

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

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

11.2.2 Printing Logs

Logging with printf

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

Custom log function

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

Ad esempio:

```
void my_log_cb(lv_log_level_t level, const char * buf)
{
  serial_send(buf, strlen(buf));
}

...


lv_log_register_print_cb(my_log_cb);
```

11.2.3 Add Logs

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

- `LV_LOG_TRACE/INFO/WARN/ERROR/USER(text)` append the following information to your `text`
- Log Level
- __FILE__
- __LINE__
- __func__
- `LV_LOG(text)` is similar to `LV_LOG_USER` but has no extra information attached.

11.2.4 API

lv_log.h

11.3 Profiler

As the complexity of the application increases, performance issues such as low FPS and frequent cache misses causing lag may arise. LVGL has internally set up some hooks for performance measurement to help developers analyze and locate performance issues.

11.3.1 Introduzione

LVGL has a built-in trace system to track and record the timestamps of important events that occur during runtime, such as rendering events and user input events. These event timestamps serve as important metrics for performance analysis.

The trace system has a configurable record buffer that stores the names of event functions and their timestamps. When the buffer is full, the trace system prints the log information through the provided user interface.

The output trace logs are formatted according to Android's systrace format and can be visualized using Perfetto.

11.3.2 Utilizzo

Configure profiler

To enable the profiler, set *LV_USE_PROFILER* in `lv_conf.h` and configure the following options:

1. Enable the built-in profiler functionality by setting `LV_USE_PROFILER_BUILTIN`.
2. Buffer configuration: Set the value of `LV_PROFILER_BUILTIN_BUF_SIZE` to configure the buffer size. A larger buffer can store more trace event information, reducing interference with rendering. However, it also results in higher memory consumption.
3. Timestamp configuration: LVGL uses the *lv_tick_get()* function with a precision of 1ms by default to obtain timestamps when events occur. Therefore, it cannot accurately measure intervals below 1ms. If your system environment can provide higher precision (e.g., 1us), you can configure the profiler as follows:

- Recommended configuration in **UNIX** environments:

```
#include <sys/syscall.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>

static uint64_t my_get_tick_cb(void)
{
    struct timespec ts;
    clock_gettime(CLOCK_MONOTONIC, &ts);
    return ts.tv_sec * 1000000000 + ts.tv_nsec;
}

static int my_get_tid_cb(void)
{
    return (int)syscall(SYS_gettid);
}

static int my_get_cpu_cb(void)
{
    int cpu_id = 0;
    syscall(SYS_getcpu, &cpu_id, NULL);
    return cpu_id;
}

void my_profiler_init(void)
{
    lv_profiler_builtin_config_t config;
    lv_profiler_builtin_config_init(&config);
    config.tick_per_sec = 1000000000; /* One second is equal␣
↪to 1000000000 nanoseconds */
    config.tick_get_cb = my_get_tick_cb;
    config.tid_get_cb = my_get_tid_cb;
    config.cpu_get_cb = my_get_cpu_cb;
    lv_profiler_builtin_init(&config);
}
```

- Recommended configuration in **Arduino** environments:

```
static uint64_t my_get_tick_cb(void)
{
    /* Use the microsecond time stamp provided by Arduino */
    return micros();
```

(continues on next page)

(continua dalla pagina precedente)

```
}

void my_profiler_init(void)
{
    lv_profiler_builtin_config_t config;
    lv_profiler_builtin_config_init(&config);
    config.tick_per_sec = 1000000; /* One second is equal to␣
↪1000000 microseconds */
    config.tick_get_cb = my_get_tick_cb;
    lv_profiler_builtin_init(&config);
}
```

4. Log output configuration: LVGL uses the `LV_LOG()` interface by default to output trace information. If you want to use another interface to output log information (e.g., file stream), you can redirect the log output using the following code:

```
static void my_log_print_cb(const char * buf)
{
    printf("%s", buf);
}

void my_profiler_init(void)
{
    lv_profiler_builtin_config_t config;
    lv_profiler_builtin_config_init(&config);
    ... /* other configurations */
    config.flush_cb = my_log_print_cb;
    lv_profiler_builtin_init(&config);
}
```

Run the test scenario

Run the UI scenario that you want to measure, such as scrolling a scrollable page up and down or entering/exiting an application.

Process the logs

Save the output log as `my_trace.txt`, use `trace_filter.py` for filtering and preprocessing:

```
./lvgl/scripts/trace_filter.py my_trace.txt
```

or

```
python3 ./lvgl/scripts/trace_filter.py my_trace.txt
```

You will obtain a processed text file named `trace.systrace`, which roughly contains the following content:

```
# tracer: nop
#
LVGL-1 [0] 2892.002993: tracing_mark_write: B|1|lv_timer_handler
LVGL-1 [0] 2892.002993: tracing_mark_write: B|1|_lv_display_refr_
↪timer
LVGL-1 [0] 2892.003459: tracing_mark_write: B|1|refr_invalid_areas
LVGL-1 [0] 2892.003461: tracing_mark_write: B|1|lv_draw_rect
LVGL-1 [0] 2892.003550: tracing_mark_write: E|1|lv_draw_rect
LVGL-1 [0] 2892.003552: tracing_mark_write: B|1|lv_draw_rect
LVGL-1 [0] 2892.003556: tracing_mark_write: E|1|lv_draw_rect
```

(continues on next page)

(continua dalla pagina precedente)

```
LVGL-1 [0] 2892.003560: tracing_mark_write: B|1|lv_draw_rect
LVGL-1 [0] 2892.003573: tracing_mark_write: E|1|lv_draw_rect
...
```

Import the processed `trace.systrace` file into Perfetto and wait for it to be parsed.

Performance analysis

If the log parsing is successful, you will see the following screen:

In the Perfetto UI, use the `A` or `D` keys to pan the timeline horizontally and the `W` or `S` keys to zoom in or out on the timeline. Use the mouse to move the focus and click on functions on the timeline to observe their execution time.

11.3.3 Add Measurement Point

Users can add their own measured functions:

```
void my_function_1(void)
{
    LV_PROFILER_BEGIN;
    do_something();
    LV_PROFILER_END;
}

void my_function_2(void)
{
    LV_PROFILER_BEGIN_TAG("do_something_1");
    do_something_1();
    LV_PROFILER_END_TAG("do_something_1");

    LV_PROFILER_BEGIN_TAG("do_something_2");
    do_something_2();
    LV_PROFILER_END_TAG("do_something_2");
}
```

11.3.4 Custom profiler implementation

If you wish to use a profiler method provided by your operating system, you can modify the following configurations in `lv_conf.h`:

- `LV_PROFILER_INCLUDE`: Provides a header file for the profiler function.
- *`LV_PROFILER_BEGIN`*: Profiler start point function.
- *`LV_PROFILER_END`*: Profiler end point function.
- *`LV_PROFILER_BEGIN_TAG`*: Profiler start point function with custom tag.
- *`LV_PROFILER_END_TAG`*: Profiler end point function with custom tag.

Taking NuttX RTOS as an example:

```
#define LV_PROFILER_INCLUDE "nuttx/sched_note.h"
#define LV_PROFILER_BEGIN          sched_note_begin(NOTE_TAG_ALWAYS)
#define LV_PROFILER_END            sched_note_end(NOTE_TAG_ALWAYS)
#define LV_PROFILER_BEGIN_TAG(str) sched_note_beginex(NOTE_TAG_ALWAYS, str)
#define LV_PROFILER_END_TAG(str)   sched_note_endex(NOTE_TAG_ALWAYS, str)
```

11.3.5 FAQ

Perfetto log parsing fails

Please check the completeness of the logs. If the logs are incomplete, it may be due to the following reasons:

1. Serial port reception errors caused by a high baud rate. You need to reduce the baud rate.
2. Data corruption caused by other thread logs inserted during the printing of trace logs. You need to disable the log output of other threads or refer to the configuration above to use a separate log output interface.
3. Make sure that the string passed in by *`LV_PROFILER_BEGIN_TAG`* or *`LV_PROFILER_END_TAG`* is not a local variable on the stack or a string in shared memory, because currently only the string address is recorded and the content is not copied.

Function execution time displayed as 0s in Perfetto

If the function execution time is lower than the precision of the timestamps, this situation can occur. You can refer to the configuration instructions above to use a higher precision timestamp.

Significant stuttering occurs during profiling

When the buffer used to store trace events becomes full, the profiler will output all the data in the buffer, which can cause UI blocking and stuttering during the output. You can optimize this by taking the following measures:

1. Increase the value of `LV_PROFILER_BUILTIN_BUF_SIZE`. A larger buffer can reduce the frequency of log printing, but it also consumes more memory.
2. Optimize the execution time of log printing functions, such as increasing the serial port baud rate or improving file writing speed.

Trace logs are not being output

If the trace logs are not automatically printed when the buffer is not full, you can try the following methods to force the log output:

1. Reduce the value of `LV_PROFILER_BUILTIN_BUF_SIZE` to fill the buffer more quickly and trigger automatic printing.
2. Manually call or use a timer to call the `lv_profiler_builtin_flush()` function to force the log output.

11.4 VG-Lite Simulator

LVGL integra un simulatore VG-Lite basato su ThorVG. Its purpose is to simplify the debugging of VG-Lite adaptation and reduce the time spent debugging and locating problems on hardware devices.

It has been integrated into the CI automated compilation and testing process to ensure that the VG-Lite rendering backend can be fully tested after each PR modification.

11.4.1 How It Works

Sort out the APIs in the `vg_lite.h` header file provided by the vendor, re-implement the APIs using ThorVG, and simulate the same rendering images as the real hardware on the simulator.

11.4.2 Configurazione

1. Enable VG-Lite rendering backend, see *GPU Generica VG-Lite*.
2. Enable ThorVG and turn on the configuration *`LV_USE_THORVG_INTERNAL`* or *`LV_USE_THORVG_EXTERNAL`*. It is recommended to use the internal ThorVG library to ensure uniform rendering results.
3. Enable *`LV_USE_VG_LITE_THORVG`* and set *`LV_DRAW_BUF_ALIGN`* to 64. The rest of the options can remain default. Make sure `LV_VG_LITE_USE_GPU_INIT` is enabled, because the thorvg drawing context needs to be initialized before it can be used.

CAPITOLO 12

Contribuire

12.1 Introduzione

Unirsi alla community di LVGL e lasciare il segno nella libreria!

Ci sono molti modi per contribuire a LVGL, anche se si è nuovi della libreria o anche solo un neofita della programmazione.

Potrebbe spaventare fare il primo passo, ma non c'è nulla di cui aver paura. Una community amichevole e disponibile ed in attesa. Si incontrano persone che la pensano in modo simile e si crea qualcosa di grande insieme.

Quindi, scopriamo quale opzione di contributo è più adatta ed è d'aiuto a partecipare allo sviluppo di LVGL!

12.2 Modi per Contribuire

- **Spargere la voce**: Condividere la propria esperienza con LVGL con gli amici o sui social media per aumentarne la visibilità.
- **Star LVGL** Dare una stella su GitHub! Aiuta molto a rendere LVGL più interessante per i nuovi arrivati.
- *Segnalare un Bug**: Aprire un GitHub Issue se qualcosa non funziona.
- **Unirsi al Nostro** Forum : Per ncontrare altri sviluppatori e porre le proprie domande.
- **Raccontare le proprie idee**: Se si ritiene che manchi qualcosa in LVGL, saremo lieti di riceverlo tramite un GitHub Issue
- *Sviluppo di Funzionalità**: Aiuta a progettare o sviluppare una funzionalità. Vedi di seguito.

Le problematiche di media e larga scala vengono discusse nelle problematiche della Feature Planning .

Una issue [problematica] può essere sviluppata quando tutte le domande nel "issue template" hanno ricevuto risposta e non ci sono obiezioni da parte di alcun membro principale.

Utilizziamo le etichette GitHub per mostrare lo stato e gli attributi delle problematiche e delle Pull Request. Per cercanre opportunità di contributo, si può Filtrare per queste etichette :

- `Simple`: Ottima scelta per iniziare con un contributo LVGL
- `PR needed`: Abbiamo esaminato il problema, ma deve ancora essere implementato
- `Review needed`: È stata aperta una Pull Request che necessita di revisione/test

12.3 Pull Request

Il merging di nuovo codice in `lvgl/lvgl` e in altri repository avviene tramite *Pull Request* (in breve PR). Una PR è una notifica del tipo "Ehi, ho apportato degli aggiornamenti al progetto. Ecco le modifiche, volendo, si possono aggiungere". Per fare ciò, c'è bisogno di una copia (chiamata fork) del progetto originale nel proprio account, apportare alcune modifiche e notificare al repository originale gli aggiornamenti. Si può vedere come appare su GitHub per LVGL qui: https://github.com/lvgl/lvgl/pulls.

Per aggiungere le modifiche, si possono modificare i file online su GitHub e inviare una nuova richiesta Pull da lì (consigliato per piccole modifiche) oppure aggiungere gli aggiornamenti nel proprio editor/IDE e usare `git` per pubblicare le modifiche (consigliato per aggiornamenti più complessi).

12.3.1 Da GitHub

1. Accedere al file da modificare.
2. Cliccare sul pulsante Edit nell'angolo in alto a destra.
3. Aggiungere le modifiche al file.
4. Aggiungere un messaggio di commit in fondo alla pagina.
5. Cliccare sul pulsante *Propose changes*.

12.3.2 Dalla Workstation Locale

Queste istruzioni descrivono il repository principale `lvgl`, ma funzionano allo stesso modo di qualsiasi repository Git remoto.

1. Eseguire il fork del repository lvgl repository. Per farlo, cliccare sul pulsante "Fork" nell'angolo in alto a destra. Il repository `lvgl` verrà "copiato" sul proprio account GitHub (`https://github.com/<YOUR_NAME>?tab=repositories`)
2. Clonare il repository forkato.
3. Aggiungere le modifiche. Si può creare un *feature branch* dal branch `master` per gli aggiornamenti: `git checkout -b <new-feature-branch-name>`
4. Eseguire il commit e il push delle modifiche nel proprio repository `lvgl` forkato.
5. Creare una PR su GitHub dalla pagina del proprio repository `lvgl` forkato (`https://github.com/<YOUR_NAME>/lvgl`) cliccando sul pulsante *"New pull request"*. Non dimenticare di selezionare il branch in cui sono state aggiunte le modifiche.
6. Impostare il branch di base in cui si desidera unire l'aggiornamento. Nel repository `lvgl`, sia le correzioni che le nuove funzionalità devono essere indirizzate al branch `master`.
7. Descrivere il contenuto dell'aggiornamento. Se applicabile, è benvenuto un codice di esempio.
8. Per apportare ulteriori modifiche, aggiornare il proprio repository `lvgl` forkato con nuovi commit. Appariranno automaticamente nella PR.

12.3.3 Formato dei Messaggi di Commit

Il formato dei messaggi di commit si ispira al Angular Commit Format.

È necessario utilizzare la seguente struttura:

```
<type>(<scope>): <subject>
            <--- blank line
<body>
            <--- blank line
<footer>
```

Possibili `<type>`:

- `fix` correzione di bug nel codice sorgente LVGL
- `feat` nuova funzionalità
- `arch` modifiche architetturali
- `perf` modifiche che influiscono sulle prestazioni
- `example` qualsiasi elemento relativo agli esempi (inclusi fix e nuovi esempi)
- `docs` qualsiasi elemento relativo alla documentazione (inclusi fix, formattazione e nuove pagine)
- `test` qualsiasi elemento relativo ai test (test nuovi e aggiornati o azioni di CI)
- `chore` qualsiasi modifica minore di formattazione o stile che renderebbe il changelog poco chiaro

`<scope>` è il nome del modulo, file o sottosistema interessato dal commit. Di solito è una sola parola e può essere scelta liberamente. Ad esempio `img`, `layout`, `txt`, `anim`. Lo scope può essere omesso.

`<subject>` contiene una breve descrizione della modifica seguendo queste linee guida:

- usare il modo imperativo: ad esempio, il presente "change", non "changed" né "changes";
- non usare la maiuscola per la prima lettera;
- nessun punto (`.`) alla fine;
- massimo 90 caratteri.

`<body>` è facoltativo e può essere utilizzato per descrivere i dettagli di questa modifica.

`<footer>` deve contenere:

- iniziare con "BREAKING CHANGE" se le modifiche 'rompono' l'API;
- riferimento al problema di GitHub o alla Pull Request, se applicabile. (Per i dettagli, vedere Linking a pull rquest to an issue.)

Alcuni esempi:

```
fix(image): update size when a new source is set
```

```
fix(bar): fix memory leak

The animations weren't deleted in the destructor.

Fixes: #1234
```

```
feat: add span widget

The span widget allows mixing different font sizes, colors and styles.
It's similar to HTML <span>
```

```
docs(porting): fix typo
```

12.4 Developer Certification of Origin (DCO)

12.4.1 Panoramica

Per garantire che tutti i criteri di licenza siano soddisfatti per ogni repository del progetto LVGL, applichiamo un processo chiamato DCO (Developer's Certificate of Origin [Certificato di Origine dello Sviluppatore]).

Il testo del DCO è disponibile qui: https://developercertificate.org/.

Contribuendo a qualsiasi repository del progetto LVGL, si accetta che il contributo sia conforme al DCO.

Se il contributo soddisfa i requisiti del DCO, non sono necessarie ulteriori azioni. In caso di dubbi, non esitare a contattarci in un commento, ad esempio nella *Pull Request* inviata.

12.4.2 Licenze Accettate e Note di Copyright

Per rendere il DCO più facile da comprendere, ecco alcune guide pratiche su casi specifici:

Il proprio lavoro

Il caso più semplice è quando il contributo è esclusivamente frutto del proprio lavoro. In questo caso si può semplicemente inviare una Pull Request senza preoccuparsi di problemi di licenza.

Utilizzo di codice da un sorgente online

Se il codice da aggiungere è basato su un articolo, un post o un commento su un sito web (ad esempio StackOverflow), è necessario seguire la licenza e/o le regole di quel sito.

Ad esempio, nel caso di StackOverflow, è possibile utilizzare una nota come questa:

```
/* The original version of this code-snippet was published on␣
↪StackOverflow.
 * Post: http://stackoverflow.com/questions/12345
 * Author: http://stackoverflow.com/users/12345/username
 * The following parts of the snippet were changed:
 * - Check this or that
 * - Optimize performance here and there
 */
 ... code snippet here ...
```

Utilizzo di codice con licenza MIT

Poiché LVGL è con licenza MIT, altro codice con licenza MIT può essere integrato senza problemi. La licenza MIT richiede l'aggiunta di una nota di copyright all'opera derivata. Qualsiasi opera derivata basata su codice con licenza MIT deve copiare il file di licenza o il testo dell'opera originale.

Utilizzare codice con licenza GPL

La licenza GPL non è compatibile con la licenza MIT. Pertanto, LVGL non può accettare codice con licenza GPL.

12.5 Stile di Codifica

12.5.1 File Template

Usare misc/lv_templ.c e misc/lv_templ.h

12.5.2 Convenzioni sui Nomi

- Le parole sono separate da '_'
- Nei nomi di variabili e di funzioni utilizzare solo lettere minuscole (ad esempio *height_tmp*)
- Negli enum e nelle definizioni utilizzare solo lettere maiuscole (ad esempio *MAX_LINE_NUM*)
- Nomi globali (API):
  - iniziano con *lv*
  - seguiti dal nome del modulo: *button*, *label*, *style* ecc.
  - seguiti dall'azione (per le funzioni): *set*, *get*, ecc.
  - chiuso col soggetto: *name*, *size*, *state* ecc.

- Typedef
  - preferire `typedef struct` e `typedef enum` invece di `struct name` e `enum name`
  - `typedef struct` e `typedef enum` terminano sempre con `_t`
- Abbreviazioni:
  - Sono utilizzate e consentite le seguenti abbreviazioni:
    - `dsc` descrittore
    - `param` parametro
    - `indev` device di input
    - `anim` animazione
    - `buf` buffer
    - `str` stringa
    - `min/max` minimo/massimo
    - `alloc` allocazione
    - `ctrl` control
    - `pos` posizione
  - Evitare di aggiungere nuove abbreviazioni

12.5.3 Guida alla Codifica

- Editor:
  - Impostare l'editor in modo che utilizzi 4 spazi per le rientranze di tabulazione (invece dei caratteri di tabulazione).
  - Eccezione: il file **Kconfig** e tutti i file make richiedono caratteri di tabulazione iniziali sugli elementi figli.
- Funzioni:
  - Scrivere funzioni che utilizzano il principio di singola responsabilità.
  - Rendere le funzioni `static` quando non fanno parte dell'API pubblica di quell'oggetto (ove possibile).
- Variabili:
  - Una riga, una dichiarazione (BAD: char x, y;).
  - Usare `<stdint.h>` (*uint8_t*, *int32_t* ecc).
  - Dichiarare le variabili dove necessario (non tutte all'inizio della funzione).
  - Utilizzare lo scope più piccolo richiesto.
  - Le variabili in un file (al di fuori delle funzioni) sono sempre *static*.
  - Non utilizzare variabili globali (utilizzare funzioni per impostare/ottenere variabili statiche).

12.5.4 Commenti

Prima di ogni prototipo di funzione nei file `.h` , includere un commento formattato in Doxygen come questo:

```
/**
 * Brief description.  Add a blank line + additional paragraphs when more␣
↪detail is needed.
 * @param  parent     brief description of argument.  Additional detail␣
↪can appear
```

(continues on next page)

(continua dalla pagina precedente)

```
 *                    on subsequent lines.  List of accepted values:
 *                    - value one
 *                    - value two
 *                    - value three
 * @return   brief description of return value.
 */
type_name_t * lv_function_name(lv_obj_t * parent);
```

Il normale prefisso di commento `/**` fa sì che il commento documenti il pezzo di codice *dopo* il commento. Quando si documenta un pezzo di codice che si trova *prima* del commento, come un membro di una struttura, usare `/**<` in questo modo:

```
/**
 * Brief description of struct
 *
 * When more detail is needed, add a blank line then the detail.
 */
typedef struct {
    char      *text;    /**< Brief description of this member */
    uint16_t   length;  /**< Brief description of this member */
} lv_example_type_t;
```

- Quando si commenta il codice, usare commenti a blocchi come questo `/* Description */`, non commenti di fine riga come questo `// Description`.
- Includi uno spazio dopo `/*` o `/**<` e prima di `*/` per migliorare la leggibilità.
- Scrivere codice leggibile per evitare commenti descrittivi come: `x++; /* Add 1 to x */`.
- Il codice dovrebbe mostrare chiaramente cosa si sta facendo.
- Si dovrebbe scrivere il **perché** è stato fatto: `x++; /* Point to closing '\0' of string */`
- Sono accettati brevi "riassunti" di poche righe: `/* Calculate new coordinates */`
- Nei commenti, utilizzare le virgolette inverse (`...`) quando si fa riferimento a un elemento del codice, come una variabile, un tipo o il nome di una struttura: `/* Update value of `x_act` */`
- Quando si aggiungono o modificano commenti, le priorità sono (in ordine di importanza):
  1. chiarezza (la facilità con cui altri programmatori possono comprendere le vostre intenzioni),
  2. leggibilità (la facilità con cui altri programmatori possono leggere i vostri commenti),
  3. brevità (la qualità di usare poche parole quando si parla o si scrive).
- Le righe vuote all'interno dei commenti sono auspicabili quando migliorano la chiarezza e la leggibilità.
- Da ricordare, quando si scrive codice sorgente, non si sta solo insegnando al computer cosa fare, ma anche insegnando ad altri programmatori come funziona, non solo agli utenti dell'API, ma anche ai futuri manutentori del codice sorgente. I commenti aggiungono informazioni su cosa si stava pensando quando è stato scritto il codice e **perché** sono state fatte le cose in quel modo, informazioni che non possono essere trasmesse dal solo codice sorgente.

Specifiche dei Commenti di Doxygen

Doxygen è il primo programma di una catena che genera la documentazione online dell'API LVGL dai file nel repository LVGL. Doxygen rileva i file a cui prestare attenzione tramite il comando `@file` all'interno di un commento Doxygen. I commenti Doxygen iniziano con `/**`. Ignora i commenti che non contengono esattamente due `*`.

Di seguito è riportata l'illustrazione di un prototipo di funzione API con documentazione che illustra la maggior parte dei comandi Doxygen utilizzati in LVGL.

```
/**
 * Set alignment of Widgets placed in containers with LV_STYLE_FLEX_FLOW␣
↪style.
 *
 * The values for the `..._place` arguments come from the `lv_flex_align_t`
 * enumeration and have the same meanings as they do for flex containers␣
↪in CSS.
 * @param  obj                  pointer to flex container.  It must have
 *                              `LV_STYLE_FLEX_FLOW` style or nothing␣
↪will happen.
 * @param  main_place           where to place items on main axis (in␣
↪their track).
 *                              (Any value of `lv_flex_align_t`.)
 * @param  cross_place          where to place item in track on cross␣
↪axis.
 *                              - `LV_FLEX_ALIGN_START`
 *                              - `LV_FLEX_ALIGN_END`
 *                              - `LV_FLEX_ALIGN_CENTER`
 * @param  track_cross_place    where to place tracks in cross direction.
 *                              (Any value of `lv_flex_align_t`.)
 * Example for a title bar layout:
 * @code{.c}
 *     lv_obj_set_flex_align(title_bar, LV_FLEX_ALIGN_START, LV_FLEX_ALIGN_
↪CENTER, LV_FLEX_ALIGN_CENTER);
 * @endcode
 * @see
 *     - https://css-tricks.com/snippets/css/a-guide-to-flexbox/
 *     - see  `lv_obj_set_flex_grow()` for additional information.
 */
void lv_obj_set_flex_align(lv_obj_t * widget, lv_flex_align_t main_place,␣
↪lv_flex_align_t cross_place,
                           lv_flex_align_t track_cross_place);
```

- Iniziare sempre un commento Doxygen con una breve descrizione dell'elemento di codice che documenta.
- Quando sono necessari maggiori dettagli, aggiungere una riga vuota sotto la breve descrizione e aggiungere ulteriori informazioni che potrebbero essere necessarie agli utenti dell'API LVGL, incluse le precondizioni per la chiamata della funzione. Doxygen necessita della riga vuota per separare le parti "brevi" da quelle di "dettaglio" della descrizione.
- Descrivere i parametri della funzione col comando `@param`. Quando una funzione scrive all'indirizzo contenuto da un parametro puntatore, se non è già ovvio (ad esempio, quando il nome del parametro contiene la parola "out"), includere la direzione nel comando per chiarezza:

  `@param[out] param_name descrizione`.
- Descrivere i valori restituiti col comando `@return`.
- Aggiungere almeno 2 spazi dopo i comandi Doxygen per una migliore leggibilità.
- Utilizzare le virgolette inverse (`...`) attorno agli elementi del codice (variabili, nomi di tipo, nomi di funzione). Per i nomi di tipo e di funzione, Doxygen genera un collegamento ipertestuale alla documentazione di quel membro del codice (quando esiste), con o senza le virgolette inverse singole.
- Aggiungere un "()" vuoto ai nomi di funzione. Doxygen non genererà un collegamento ipertestuale alla documentazione della funzione senza questo.
- Utilizzare una grammatica corretta per chiarezza. Le descrizioni dei parametri non necessitano di punti dopo di esse, a meno che non siano frasi complete.
- Allineare i bordi del testo attorno agli elenchi di parametri per facilitarne la lettura.

- Gli elenchi (ad esempio, dei valori dei parametri accettati) possono essere creati utilizzando il carattere '-'. Se l'elenco deve essere numerato, è possibile utilizzare anche i numeri.
- Inserire il codice di esempio in un blocco di codice circondandolo con i comandi `@code{.c}` e `@endcode`. Doxygen non necessita della parte `{.c}`, ma il software a valle sì.
- Indirizzare il lettore a informazioni aggiuntive utilizzando il comando `@see`. Doxygen aggiunge un paragrafo "See also" [Vedi anche]. Il testo che segue il comando `@see` verrà indentato.
- Se si crea una nuova coppia di file `.c` e `.h` (ad esempio per un nuovo driver), assicurarsi che un commento Doxygen come questo sia presente all'inizio di ogni nuovo file. Doxygen non analizzerà il file senza il comando `@file`.

```
/**
 * @file filename.c
 *
 */
```

Comandi Doxygen Supportati

- `@file` indica a Doxygen di analizzare questo file e, se applicabile, fornisce anche la documentazione sul file stesso (tutto ciò che lo segue nello stesso commento).
- `@param name description` documenta `name` come parametro di funzione, e `description` è il testo che lo segue finché Doxygen non incontra una riga vuota o un altro comando Doxygen.
- `@return description` documenta il valore di ritorno finché Doxygen non incontra una riga vuota o un altro comando Doxygen.
- `@code{.c}/@endcode` racchiude il codice che deve essere inserito in un blocco di codice. Mentre Doxygen sa utilizzare la codifica a colori C per i blocchi di codice in un file .C, la parte a valle della sequenza di generazione della documentazione non lo fa, quindi l'aggiunta di `{.c}` al comando `@code` è necessaria.
- `@note text` inizia un paragrafo in cui è possibile inserire una nota. La nota termina con una riga vuota, la fine del commento o un altro comando Doxygen che inizia una nuova sezione. Se la nota contiene più di un paragrafo, è possibile aggiungerne altri utilizzando altri comandi `@note`. Al momento della stesura di questo documento, i comandi `@par` non aggiungono paragrafi aggiuntivi alle note, come indicato nella documentazione di Doxygen.
- `@see text` genera un paragrafo "See also" [Vedi anche] in una sezione evidenziata, utile quando si trovano altrove delle informazioni aggiuntive su un argomento.

12.5.5 Convenzioni API

Per supportare la generazione automatica dei binding, l'API C LVGL deve seguire alcune convenzioni di codifica:

- Utilizzare `enum` invece delle macro. Se è inevitabile l'uso dei `define` esportarli con `LV_EXPORT_CONST_INT(defined_value)` subito dopo il `define`.
- Negli argomenti delle funzioni, utilizzare la dichiarazione `type name[]` per i parametri dell'array invece di `type * name`
- Utilizzare puntatori tipizzati invece di puntatori `void*`
- Il costruttore del widget deve seguire il pattern `lv_<widget_name>_create(lv_obj_t * parent)`.
- Le funzioni membro del widget devono iniziare con `lv_<widget_name>` e devono ricevere `lv_obj_t*` come primo argomento, che è un puntatore all'oggetto Widget stesso.
- Le API `struct` devono seguire le convenzioni dei widget. Questo significa ricevere un puntatore alla `struct` come primo argomento, e il prefisso del nome della `struct` dovrebbe essere usato anche come prefisso del nome della funzione (ad esempio `lv_display_set_default(lv_display_t * disp)`)
- Le funzioni e le `struct` che non fanno parte dell'API pubblica devono iniziare con un carattere di sottolineatura per essere contrassegnate come "private".

- Anche l'argomento deve essere nominato nei file H.
- Non usare `malloc` in una variabile statica o globale. Invece, dichiarare la variabile nella struttura `lv_global_t` in `lv_global.h` e contrassegnarla con `(LV_GLOBAL_DEFAULT()->variable)` quando viene utilizzata.
- Per registrare e utilizzare le callback, è necessario seguire una delle seguenti procedure.
  - Passare un puntatore a una `struct` come primo argomento sia della funzione di registrazione che del callback. Tale `struct` deve contenere il campo `void * user_data`.
  - L'ultimo argomento della funzione di registrazione deve essere `void * user_data` e lo stesso `user_data` deve essere passato come ultimo argomento della callback.

Per ulteriori informazioni, consultare la documentazione di MicroPython.

12.5.6 Formattazione

Ecco un esempio per mostrare il posizionamento delle parentesi e l'uso degli spazi:

```
/**
 * Set new text for a label.  Memory will be allocated by label to store␣
 ↪text.
 *
 * @param  label  pointer to label Widget
 * @param  text   '\0' terminated character string.
 *                NULL to refresh with 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 command is inline with␣
 ↪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␣
 ↪starts on same line */
        lv_label_refr_text(label);
        return;
    }

    ...
}
```

È possibile utilizzare **astyle** per formattare il codice. Eseguire `code-format.py` dalla cartella `scripts`.

12.5.7 Include

Vari sottosistemi di LVGL possono essere abilitati o disabilitati impostando una macro in `lv_conf.h` rispettivamente su 1 o 0. I file di codice che contengono la logica per tali sottosistemi sono spesso organizzati in modo che vi sia una direttiva `#if <ENABLING_MACRO_NAME>` all'inizio del file e la sua corrispondente `#endif` alla fine. Se si aggiunge o si modifica un sottosistema di questo tipo in LVGL, ove possibile, l'unica `#include` che dovrebbe trovarsi sopra tali direttive condizionali dovrebbe essere sufficiente a includere la macro di abilitazione/disabilitazione. In particolare:

- nel file `.c`: l'`#include` che include l'header con la corrispondenza più vicina a quel file `.c`
- nel file `.h` corrispondente: `#include "lv_conf_internal.h"`

che, a sua volta, include `lv_conf.h`. Vedere gli esempi in lv_freetype.c, lv_freetype_private.h e lv_freetype.h.

12.5.8 pre-commit

pre-commit è un gestore di pacchetti multilinguaggio per gli hook di pre-commit. Consultare la installation guide per installare il pacchetto Python di pre-commit sul computer di sviluppo.

Una volta installato `pre-commit` si dovranno configurare gli script degli hook di git con:

```
pre-commit install
```

ora `pre-commit` verrà eseguito automaticamente su `git commit`!

12.5.9 Hook

L'hook locale `format-source` (vedere `.pre-commit-config.yaml`) esegue **astyle** su tutti i file sorgente e header in staging (che non sono esclusi, vedere la chiave `exclude` della configurazione dell'hook) prima di inserire il messaggio di commit. Se un file viene formattato da **astyle** sarà necessario aggiungere la modifica all'area di staging ed eseguire nuovamente `git commit`.

L'hook `trailing-whitespace` corregge gli spazi vuoti finali su tutti i file.

12.5.10 Saltare gli hook

Si può saltare un hook in particolare, con:

```
SKIP=name-of-the-hook git commit
```

12.5.11 Hook di test

Non è necessario eseguire un commit per testare gli hook, si possono testare aggiungendo i file nell'area di staging ed eseguendo:

```
pre-commit run name-of-the-hook
```

CAPITOLO 13

Change Log

13.1 v9.3.0 3 June 2025

13.1.1 Summary

This is a significant release packed with exciting updates from both our community and internal team.

One of the most notable additions is **XML support**, enabling you to describe your UI in a declarative manner. This way LVGL UI can be loaded at runtime without recompiling and deploying the application. This feature is also a key requirement for the upcoming UI Editor we're developing.

The documentation has been thoroughly reorganized and proofread.

We've also laid the groundwork to support **3D textures** as LVGL widgets.

To support our partners, we've added numerous new drivers and documentation:

- STM's DMA2D
- STM's NeoChrom GPU
- STM's LTDC LCD peripheral
- Generic NemaGFX driver
- NXP's G2D GPU
- UEFI BIOS driver
- Toradex documentation
- Torizon OS guide
- Buildroot integration
- Added dmabuf support to the Wayland driver (currently limited to G2D draw unit)
- Hotplug support in evdev with auto-discovery
- Optional use of Linux CPU usage statistics

Text rendering has become much more advanced:

- Our FreeType binding now supports colored glyphs and glyph outlines
- Text recoloring is back (e.g., `"A #ff0000 red# word"`)

- Improved GPU integration to render A1/A2/A4 bitmaps directly

And finally, some other important additions: - New color formats: `RGB565_SWAPPED` (commonly used with SPI-based display controllers) and `ARGB8888_REMULTIPLED` (required for Wayland and Lottie animations) - SVG support - Double and triple click detection - Tiled rendering to better utilize multiple CPU cores - Multi-touch gestures (swipe, pinch, rotate) - Global recolor style property (tint all widgets and images) - Triple buffer support

13.1.2 Architectural

- **arch(font_manager): add multiple font backend support** 8038
- **arch(demos): move new demos to lv_demos** 8022
- **arch(cmake): add native Kconfig support to cmake** 7934

13.1.3 New Features

- **feat(wayland): add dmabuf support with g2d** 8122
- **feat(draw_sw): add RGB565_SWAPPED support** 8227
- **feat: add new lvgl examples** 8225
- **feat(barcode): support raw code 128** 8287
- **feat(draw_opengles): add getter fn for glfw_window and fix freeing non allocated textures** 8257
- **feat(draw): add configurable thread priority for all drawing units** 8162
- **feat(test): add Dockerfile support with CI env** 8209
- **feat(image): support symbol images with inner alignment** 8182
- **feat(disp): allow rotation with `FULL` render mode** 8107
- **feat(display): add triple buffer support** 8158
- **feat(svg): add API for getting original width and height** 8180
- **feat(cache): add new `lv_cache_lru_ll` module** 8155
- **feat(cmake): disable PCPP by default** 8126
- **feat(animimage): support set source interfaces with or without parameter of reverse play** 8164
- **feat(drivers): add ft81x framebuffer driver** 7815
- **feat(animimage): support images play in a reversed order** 8085
- **feat(font): Replace SimSun font with SourceHanSansSC** 8006
- **feat(widget): Add `lv_3dtexture` widget and 3D draw task type** 8033
- **feat(nuttx): auto enable display matrix rotation** 8119
- **feat(refr): use transform matrix to realize display rotation** 6911
- **feat(image): add LV_IMAGE_ALIGN_CONTAIN & LV_IMAGE_ALIGN_COVER to scale images without changing aspect ratio** 7955
- **feat(sdl): add window icon settings** 7808
- **feat(sw_blend) : add argb8888-premultiplied support** 7979
- **feat(linux/fbdev): support non-mmappable frame buffers** 8058
- **feat(disp): support subscription and unsubscription of vsync event** 7487
- **feat(demos): add WiFi credentials input to the high res demo** 7953
- **feat(scale): add id1 id2 for tick line draw descriptors. fix zero division** 7695
- **feat(scripts): Add --name parameter to LVGLImage.py** 7996

- **feat(draw/sw): add support for vector fonts** 7560
- **feat(draw): add comments and unify struct names** 7878
- **feat(refr): add global recolor** 7855
- **feat(lz4): update lz4 to 0.10.0** 7869
- **feat(test): add test cases for indev_gesture pinch** 7947
- **feat(chart): implement get_index_from_x() for LV_CHART_TYPE_SCATTER** 7824
- **feat(demo): New smartwatch demo** 7883
- **feat(indev): Add rotation and two fingers swipe gestures support** 7865
- **feat(obj_tree): indent printed object data according to its depth** 7852
- **feat(draw_sw): have only one SW draw unit with multiple threads internally** 7899
- **feat(dma2d): add support for DMA2D on STM32H7RS** 7850
- **feat(benchmark): add on benchmark end callback** 7814
- **feat(tests): add draw 8bpp font test** 7895
- **feat(spangroup): add some testcases for span rtl mode.** 7870
- **feat(examples): add looping scroll example** 7714
- **feat(draw/sw): allow custom handlers** 7531
- **feat(nxp): add G2D support** 7625
- **feat(lv_bin_decoder): improve error logging in file reading** 7826
- **feat(blend): add blend difference mode** 7796
- **feat(thorvg): use LVGL's malloc/realloc/zalloc/free** 7772
- **feat(nuttx_image_cache): add configuration to use image cache heap for default heap** 7653
- **feat(vg_lite): optimize resource reference count management** 7809
- **feat(mem): add lv_reallocf** 7780
- **feat(span): support bidi for span** 7717
- **feat(draw_vector): add fill units support** 7774
- **feat(docs): new docs-build paradigm...** 7597
- **feat(docs): upgrade presentation of display.rst** 7478
- **feat(decoder): add svg image decoder for image widget** 7141
- **feat(osal): add lv_os_get_idle_percent for linux** 7632
- **feat(fs): clarify rest of docs on driver-identifier letters.** 7710
- **feat(vg_lite): enhanced the automatic diagnosis function of GPU errors** 7751
- **feat(demos): add new card to high resolution demo** 7699
- **feat(observer): add bind_XXX ge/gt/le/lt and notify only when value changes** 7678
- **feat(drm): add support for GBM buffer object to increase performance** 7464
- **feat(draw/sw): make I1 luminance threshold configurable** 7616
- **feat(drivers): add evdev discovery** 7481
- **feat(docs): proofread/edit obj_property.rst.** 7621
- **feat(demo): add smartwatch demo** 7429
- **feat(scripts/LVGLImage.py): adds RGB565 dithering support** 7582

- **feat(draw): add lv_draw_letter support** 7490
- **feat(anim): add a pause method** 7583
- **feat(docs): batch 14 of proofread/edited docs** 7477
- **feat(docs): add short bit about include pattern to CODING_STYLE.rst** 7563
- **feat(docs): batch 12 of proofread/edited docs** 7440
- **feat(dma2d): add support for ARGB1555 color on top of RGB565 format** 7555
- **feat(drivers): add UEFI driver** 7069
- **feat(demos): High Resolution Demo Improvements** 7566
- **feat(xml): add support for more properties and add more examples** 7417
- **feat(libinput): map LV_KEY_{ESC}** 7544
- **feat(docs): batch 11 of proofread/edited docs** 7361
- **feat(docs): batch 10 of proofread/edited docs** 7357
- **feat(docs): batch 13 of proofread docs** 7458
- **feat(dropdown): add lv_anim_enable_t parameter to lv_dropddown_set_selected** 7310
- **feat(demos): add high resolution demo** 7308
- **feat(draw/sw): added support for 3 bpp font rendering** 7350
- **feat(gen_json): adds option to gen_json to build without docstrings** 7471
- **feat(docs): make docs development practical** 7414
- **feat(osal): add SDL2 based threading and OS support** 7457
- **feat(anim): clarify reverse play in animation API** 7338
- **feat(draw): add layer memory allocation config support** 7038
- **feat(NemaGFX): add freetype vector font support** 7346
- **feat(refr): improve performance measurement** 7430
- **feat(display): add draw buffer size getter** 7332
- **feat(docs): proofread/edit batch 9** 7324
- **feat(docs): widget proofread wrap-up** 7405
- **feat(docs): update threading details in answer to #7313** 7342
- **feat(profiler_builtin): support nanosecond accuracy** 7415
- **feat(vg_lite): reduce unnecessary path quality settings** 7398
- **feat(examples): add infinite scroll example** 7388
- **feat(docs): batch 8 of proofread/edited docs** 7295
- **feat(docs): document LV_DPX(n)** 7374
- **feat(file_explorer): remove '.' and rename '..' to '< Back'** 7270
- **feat(sdl): add I1 color format render support** 7036
- **feat(drivers/st_ltdc): add rotation support to LTDC driver** 7254
- **feat(chart,calendar): two grammar corrections** 7340
- **feat(scroll): user-defined scrollbar length using LV_STYLE_LENGTH** 7306
- **feat(draw_sw): add image clip_radius and mask before transformation** 7244
- **feat(doc): add documentation on Torizon OS** 7280

- **feat(docs): batch 6 of proofread/edited docs** 7277
- **feat(scroll): adjust scroll behavior for non-elastic objects** 7336
- **feat(code-format): minor enhancements** 7311
- **feat(font): allow using A1,2,4 bitmaps + handle byte aligned fonts** 7234
- **feat(fsdrv): set the working directory** 7272
- **feat(observer): add subject snprintf** 7250
- **feat(indev): add multi touch gestures** 7078
- **feat(docs): batch 7 of proofread/edited docs** 7281
- **feat(png): add support for files without extension** 7289
- **feat(docs): scrolling doc proofread, edited and clarified...** 7170
- **feat(issue): add platform description to bug-report** 7273
- **feat(dropdown): add animations on rotary event** 7271
- **feat(docs): batch 5 of proofread/edited docs** 7218
- **feat(indev): add setter for long press repeat time** 7235
- **feat(docs): buttonmatrix proofread/edit** 7194
- **feat(obj): add transform matrix attribute** 7187
- **feat(sdl): use SDL_Delay for delay callback** 7243
- **feat(docs): batch 4 of proofread/edited docs** 7207
- **feat(observer): add null pointer check** 7183
- **feat(docs): plea for proper word-wrapping in `.rst` files.** 7189
- **feat(thorvg): update thorvg to 0.15.3** 7103
- **feat(docs): batch 3 of proofread/edited docs** 7180
- **feat(sw): Add method to convert a htiled I1 buffer to vtiled** 7129
- **feat(fs_posix): add error code conversion** 7166
- **feat(roller): set roller option with a string** 7143
- **feat(docs): proofread and edited docs** 7144
- **feat(draw_label): Support simultaneous text selection and recolor** 7116
- **feat(font): support 8 bpp font bitmaps** 7100
- **feat(vg_lite): add more detailed error dump information** 7104
- **feat(ffmpeg): add playback complete event trigger** 7119
- **feat(docs): reorganize docs** 7136
- **feat(scale): add additional style properties** 6649
- **feat(vg_lite): add ARGB1555 ARGB4444 ARGB2222 support** 7028
- **feat(sdl): add float zoom window support** 7089
- **feat(scripts/gdb): add draw unit debug info support** 7095
- **feat(tests): add log print callback** 7076
- **feat(nuttx): add indev cursor display** 7021
- **feat(libs): add SVG rendering support** 6845
- **feat(drivers): add STM32 LTDC support** 7059

- **feat(demo): add an ebike demo** 7019
- **feat(opengl): texture caching** 6861
- **feat(draw): add NemaGFX rendering backend** 7002
- **feat(text): add text recolor back** 6966
- **feat(vg_lite): add profiler for vg_lite_set_scissor** 7023
- **feat(vg_lite_tvg): add vg_lite_set_scissor function support** 6959
- **feat(nxp): Release/nxp patches for LVGL master** 6978
- **feat(render): basic of tiled rendering** 6761
- **feat(sdl): speed up rotation** 6835
- **feat(nuttx): move static var index to global for gdb diagnostic** 6890
- **feat(style): add a "set all" function for margin** 6904
- **feat(gif): add loop count control** 6839
- **feat(snapshot): add argb8565 support** 6899
- **feat(array): add an empty element when element is NULL** 6893
- **feat(vg_lite): add A8 and L8 dest buffer support** 6884
- **feat(animimg): add getter function for underlying animation** 6838
- **feat(vg_lite): add yuy2 color format support** 6882
- **feat(profiler): support different module divisions** 6834
- **feat(switch): add vertical switch function** 6786
- **feat(spangroup): add `lv_spangroup_get_span_by_point` in spangroup** 6579
- **feat(indev): detect double and triple click (closes #6020)** 6187
- **feat(dma2d): add basic support** 6691
- **feat(nuttx): add memory dump for image cache heap** 6807
- **feat(vg_lite): optimize check_image_is_supported** 6802
- **feat(vg_lite): add draw border side support** 6796
- **feat(freetype): add colored glyphs support** 6686
- **feat(event): event supports delayed deletion ability** 6655
- **feat(libs): add freetype font manager** 6482
- **feat(stdlib): add lv_calloc function** 6743
- **feat(fonts) add aligned fonts for demo benchmark** 157ee6a
- **feat(xml): add the basics of declarative XML support** fc5939d
- **feat(test): make LVGL's test utilities public** 8d04466
- **feat(xml): add support scale, span, roller and bar** a535063
- **feat(circle_buff): add lv_circle_buf_t component** d08d545
- **feat(xml): add subject and global/local scoping support** 2ca425c
- **feat(xml): add buttonmatrix** 299e31b
- **feat(obj_name): add auto-indexing with names like 'mybtn_#'** e4bbc4f
- **feat(xml): add gradient support** b78a9b4
- **feat(gdb): add lvgl GDB plugin** d460edb

- **feat(xml): add basic callback event support** 1c9c9f6
- **feat(xml): add canvas and calendar support** 79381cc
- **feat(xml): add support for textarea and keyboard** b63472d
- **feat(xml): add arc and chechkbox** 3095b63
- **feat(xml): add obj, roller, dropdown, and arc bind_* properties** 82996a5
- **feat(iter): add lv_iter_t module** 4d7f577
- **feat(xml): load fonts and images from XML** 670845e
- **feat(xml): add test for a complex view** 052d908
- **feat(xml): add the first version of property updater** b23a228
- **feat(image_cache): add dump info ability for image_cache** 30f0d6c
- **feat(property): support second value in property** 133b6fc
- **feat(xml): support removing style properties** ed60b20
- **feat(freetype): add font kerning support** 0414c78
- **feat(slider): add property interface** 4eacf15
- **feat(vglite) draw_vglite_border add support for legacy scissor** 2e75bbd
- **feat(animimage): add property support** f5b00f5
- **feat(draw_sw): draw A8 static fonts directly in SW render** 102b633
- **feat(test/freetype): add vg-lite outline font test** feee9ae
- **feat(xml): handle registering assets multiple times** 910cf08
- **feat(NemaGFX): add TSC color formats** 07854b1
- **feat(slider): add orientation support (based on bar)** f23b42b
- **feat(vglite) add LV_USE_VGLITE_STATIC_BITMAP option to use static glyphs for text rendering** 7ba6cf2
- **feat(chart): add lv_chart_set_cursor_pos_x/pos_y()** 8d0a519
- **feat(nuttx_fbdev): support for complete non-/consecutive fbdev mmap processing** 340482d
- **feat(label_xml): support the format string in bind_text** c5022ce
- **feat(xml): support styles with the same name** 7f45e2f
- **feat(observer): support simple int binding to label** c23636c
- **feat(observer): rename arg to clarify correct use...** 82034cb
- **feat(xml): use LVGL's malloc for parsing** 717301a
- **feat(obj): add state processing in XML parser** 81eb192
- **feat(xml): add support for ext_click_area** d91bdc5
- **feat(label_xml): support the format string in bind_text** 3d1beab
- **feat(obj): add obj name support** 626d6b5
- **feat(observer): Make `lv_obj_remove_from_subject()` legal for Widget Binding.** 9b3c365
- **feat(lv_subject_t): re-sequence fields to reduce size from 32 => 28 bytes** e0dfe80
- **feat(indev): detect double and triple click (closes #6020) (#6187)** 651f69f

13.1.4 Performance

- **perf(blend): fix loop unrolling condition** 8306
- **perf(vg_lite): relaxing the matrix transform judgment criteria** 8219
- **perf(label): reduce the time of calling lv_text_get_size when drawing** 8129
- **perf(vg_lite): switch gradient cache lru_rb -> lru_ll** 8188
- **perf(vg_lite): add gradient pool to optimize memory allocation** 8149
- **perf(refr): reduce refr_children call by check layer->opa** 8133
- **perf(draw): support draw task dsc alloc together to reduce the malloc call times** 8070
- **perf(draw): reduce empty dispatch** 8073
- **perf(vg_lite): add font cache reference counting support** 7407
- **perf(array): make short functions inline to reduce function jumps** 7890
- **perf(vg_lite): reduce ineffective D-cache flushing operations** 7817
- **perf(draw): optimize lv_draw_label() with text_length for text_local** 7673
- **perf(obj): use layer to cache the current opa stack** 7523
- **perf(nxp/vglite): do not recreate identity matrix at runtime** 7595
- **perf(vg_lite): optimize matrix and rectangle conversion** 7537
- **perf(refr): reduce matrix conversion** 7536
- **perf(vg_lite): improve path append data performance** 7504
- **perf(bin_decoder): improve the decoding performance of a8** 7494
- **perf(vg_lite): improve path data conversion performance** 7470
- **perf(os): optimize OS API calls without OS mode** 7420
- **perf(span): optimize span render performance up by 50%** 7290
- **perf(draw): improve execute time of obj's transforming** 6638
- **perf(vg_lite): reduce redundant matrix calculations** 7163
- **perf(label): simplify handling of bytes overwritten with dots** 7001
- **perf(benchmark): use XRGB8888 image withn 32bit color depth** 6914
- **perf(vg_lite): optimize label drawing efficiency** 6853
- **perf(draw): skip area independence tests with one draw unit** 6825
- **perf(obj): skip repeated flag setting operations** 6859
- **perf(obj): reduce unnecessary border post drawing** 6852
- **perf(vg_lite): reduce matrix and radius calculations** 6800
- **perf(vg_lite): balancing performance and memory consumption** 6823
- **perf(demo): use set_text_static in demos to cache them ins SDL and OpenGL renderers** 926e3df

13.1.5 Fixes

- **fix(nema_gfx): fix STM32U5 compilation with NEMAGFX enabled and NEMAVG disabled** 8291
- **fix(vg_lite): add nullptr check for vg_dash_pattern** 8352
- **fix(example): fix params to lv_rand call** 8324
- **fix(NemaGFX): get the static bitmap even if bpp is not 8** 8347

- **fix(NemaGFX): new label static bitmap handling and fix area move bug** 8332
- **fix(esp/nuttx): add missing include** 8344
- **fix(display): fix divide by stride of zero value** 8308
- **fix(calendar): allow setting years in ascending order** 8315
- **fix(calendar): fixed macro comparing unsigned value to 0** 8316
- **fix(sysmon): define perf monitor log mode when perf monitor is not enabled** 8329
- **fix(themes): Fix mono theme init** 8343
- **fix(chart): fix variable overflow** 8318
- **fix(evdev): mark unused variable as unused** 8330
- **fix(span): fix dereference obj before checking for null** 8317
- **fix(draw_img): pass correct variable to LV_DRAW_SW_IMAGE** 8314
- **fix(draw_sw): in ARGB8888_PREMULTIPLIED fix rounding error and add RGB888 image blending support** 8264
- **fix(xml): fix tabview XML definition** 8309
- **fix(freetype_image): dereference null pointer** 8307
- **fix(theme): fix judgment logic error.** 8303
- **fix(pxp): sync rotation direction with SW render** 7063
- **fix(draw_vector): draw vector not calc draw_buf offset** 8262
- **fix(svg): correctly parse header** 8281
- **fix(vg_lite): fix strict alias warning with higher optimization levels** 8136
- **fix(tiny_ttf): fix GPOS lookup list table address** 8251
- **fix(layout): size in cross direction was not updated when layout on parent was changed from grid to flex** 8053
- **fix(wayland): prevent wayland reinitialization to support multiple windows** 8273
- **fix(wayland): assert surface is configured after creating window** 8271
- **fix(bar): fix bar indicator length error** 8200
- **fix(vg_lite): fix access to uninitialized members** 8256
- **fix(SW) build with LV_DRAW_SW_COMPLEX disabled** 8246
- **fix(vg_lite): enable box shadow by default** 8242
- **fix(dave2d): fix evaluate callback** 8253
- **fix: format specifier in lv_obj_tree.c** 8244
- **fix(refr): add missing area intersect check** 8240
- **fix(refr): fix matrix rotation precision loss** 8221
- **fix(bin_decoder): fix build warnings when decompressing LZ4 compressed images** 8234
- **fix(vg_lite): reset the scissor area when rendering with vg_lite.** 8232
- **fix(pinyin): update candidates on the second letter too** 8105
- **fix(wayland): support version 2 of the XDG protocol** 8201
- **fix(log): fix possible level out of bounds** 8216
- **fix(nxp/g2d): remove useless g2d_search_buf_map() when free buf** 8072
- **fix(wayland): Remove the XDG_RUNTIME_DIR check** 8041

- **fix(span): replace deprecated spangroup_set_mode fn from examples** 8192
- **fix(thorvg): fix incompatibility with c++20 (#8042)** 8043
- **fix(lottie): smooth edges on lottie animation** 8189
- **fix(roller): don't send click event when scrolled** 8101
- **fix(span): add assertion for bidi text buffer allocation** 8168
- **fix(display/renesas_glcdc): stride of the framebuffer can be different to the width** 8177
- **fix(dave2d): fix compilation error** 8175
- **fix(circle_buf): don't clear array on reset** 8157
- **fix(refr): change clip area don't take effect on children and draw post** 8117
- **fix(examples): fix #warning causing examples build to fail.** 8135
- **fix(list): check that LV_USE_FLEX is enabled when using LV_LIST** 8139
- **fix(opa): corrected incorrect usage of LV_OPA_MIN** 8161
- **fix(nuttx_image_cache): fix compilation issues** 8165
- **fix(NemaGFX): update STM32U5 nema lib to revC** 8138
- **fix(drivers/evdev): process pointer coordinates in unrotated frame** 8061
- **fix(vg_lite): fix vector draw pattern matrix error** 8134
- **fix(arc): handle clicks on a full circle** 8106
- **fix(cache): prevent resource leaks in cache entry creation failure** 8144
- **fix(indev): fix indev gesture occasional crash** 8146
- **fix(ffmpeg): fix ffmpeg decoder assert** 8128
- **fix(makefile): Extend component.mk with missing elements** 8118
- **fix(tabview):fix example lv_example_tabview_2** 8084
- **fix(vg_lite): fix vector drawing not handling global matrix** 8115
- **fix: var redeclaration** 8109
- **fix(cmake): fix LV_CONF_PATH quoting consistency issue** 8079
- **fix(sdl): fix access fb2 NULL pointer** 8096
- **fix: table get cell area error** 8047
- **fix(test): do not enable test_bg_image with || 1** 8081
- **fix(makefile): Path fixes in component.mk** 8068
- **fix(area): test and fix lv_area_diff edge case** 7907
- **fix(font): allow non-constant LV_FONT_DEFAULT again (fixes #7788)** 8059
- **fix(draw asm): replace attribute syntax for asm arm files** 8076
- **fix(matrix): use homogeneous coordinates to transform point** 7960
- **fix(draw): fix "blend_non_normal_pixel: Not supported blend mode" issue when using lv_demo_smartwatch which compiled by MSVC** 8017
- **fix(draw_chart): remove raw_end setting in draw_series_line** 8024
- **fix(draw_sw): fix rotation typos** 8050
- **fix(lottie): revert example** 8054
- **fix(draw asm): fix GCC linker error** 8055

- **fix(tree): add NULL check** 7972
- **fix(draw_sw): make the images invisible on full recolor to the background color** 7868
- **fix(text): handle recolor in lv_text_get_size** 8026
- **fix(observer): check if observer is associated with obj on remove fn** 7727
- **fix(example_anim): remove scrollable flag** 8008
- **fix(lv_image): fix touch area calculation** 8027
- **fix(vg_lite): check for better draw unit** 8040
- **fix(draw_sw): fix letter outline multi-threading issues** 8003
- **fix(docs): fix Riverdi & Viewe links** 8031
- **fix(draw_nema_gfx_triangle): use correct field names for lv_draw_triangle_dsc_t** 8005
- **fix(getcwd): capture return value #7991** 7992
- **fix(label): fix long mode clip #7922** 7957
- **fix(svg): adjust svg render node object for reduce memory usage.** 8013
- **fix(micropython): lvgl module deinit** 7973
- **fix(draw/sw): draw outline span wrong init.** 8011
- **fix(tests): explicitly install libudev dependency #7983** 7985
- **fix(scripts): initialize reserved fields of lv_image_dsc_t and lv_image_header_t to prevent compiler warnings** 7799
- **fix(demo): add LV_USE_LOTTIE check for demo smartwatch** 8002
- **fix(gif): free memory on error** 7950
- **fix(draw_sw_vector): fix thorvg canvas colorspace** 7975
- **fix(thorvg.h): ensure can use standard ints #7988** 7989
- **fix(color): add cast to LV_OPA_MIX macros** 7956
- **fix(Widgets): Optimise widget event callbacks to remove unnecessary calls to the event handler.** 7954
- **fix(vg_lite): modify recolor image behavior** 7977
- **fix(wayland): Ensure variable is initialized before use #7986** 7987
- **fix(bin_decoder): fix the crash when decoder A8 images and flush cache** 7952
- **fix(image_decoder): enhance code robustness** 7969
- **fix(anim): fix the crash caused by delete anim in cb** 7926
- **fix(obj): remove all events from the object** 7811
- **fix(buttonmatrix): initialize auto_free_map in constructor** 7966
- **fix(draw_sw): fix memory leaks** 7964
- **fix(nxp/g2d): fix memory management error in G2D buffer mapping** 7875
- **fix(thorvg): fix internal types of rasterXYFlip** 7782
- **fix(port_releaser): always use remote when pushing and add token** 7856
- **fix(X11): call XCloseDisplay in delete** 7904
- **fix(refr): make lv_display_refr_timer public again** 7925
- **fix(scale): remove dangling reference to stack buffer** 7915
- **fix(examples): add casts and change int types to fix checker warnings** 7933

- **fix(observer): fix lv_subject_notify_if_changed not defined when LV_USE_DROPDOWN = 0** 7783
- **fix(decoder): missing log_trace when decoders are not found.** 7790
- **fix(indev): remove redundant zero-initialization operations** 7929
- **fix(scale): change return type of lv_scale_get_rotation to int32_t and update documentation** 7862
- **fix(obj): style opa of other part is not effective** 7905
- **fix(nuttx): incorrect draw buffer size for I1 color and bpp=1** 7885
- **fix(label): fix behavior when set to `LV_LABEL_LONG_MODE_SCROLL_CIRCULAR` with BIDI** 7886
- **fix(svg): fix SVG draw rect off by 1px bug** 7902
- **fix(dave2d): LV to D2 colour format conversion for I8 format** 7896
- **fix(dave2d): fix build break** 7882
- **fix(thorvg): add missing include** 7877
- **fix(nuttx): fix nuttx lcd release assert** 7840
- **fix(refr): fix invalidate area calc error** 7871
- **fix(vg_lite): fix missed reference count release** 7889
- **fix(freetype): add missing error handling for glyph bitmap lookup failure** 7887
- **fix(vg_lite): fix pending swap sequence error** 7849
- **fix(indev): fix lv_indev_gesture write access error** 7843
- **fix(SDL): error if color depth is 1 and render mode is not partial** 7846
- **fix(draw): fix 'lv_vector_path_append_arc' not starting with a 'move to' operation.** 7854
- **fix(release_updater): push: if branch is specified, remote must be specified too** 7863
- **fix(obj): cover check should consider both grad opa-s** 7813
- **fix(indev): fix platform-specific printf format for int32_t** 7844
- **fix(profiler_builtin): fix uint32_t storage nanosecond time overflow** 7818
- **fix(demos/smartwatch): fix image color format error** 7819
- **fix(nema): Broken build after draw unit refactoring, draw_label changes** 7759
- **fix(release_updater): minor fixes and addition of new boards** 7726
- **fix(examples): add casts** 7831
- **fix(indev): fix scroll_obj not send LV_EVENT_INDEV_RESET** 7767
- **fix(kconfig): add LIBUV dependency to prevent missing configurations** 7798
- **fix(indev): modify calculation delta_y use p_delta_y, not p_delta_x** 7791
- **fix(example_grad): return value from lv_style_get_prop not checked** 7793
- **fix(indev): fix platform-specific printf format for int32_t** 7784
- **fix(draw_letter): fix draw letter bg_coords behaviour when it's NULL** 7773
- **fix(text): fix oob read for utf8-next** 7602
- **fix(bin_decoder): check divisor is zero** 7610
- **fix(nema): fix indexed image error** 7744
- **fix(DMA2D): build issue after the draw_unit to draw_task refactor** 7749
- **fix(vg_lite): alleviate the loss of precision in obtaining path bound** 7731

- **fix(indev): add the missing wait_until_release flag to clean up** 7638
- **fix(draw): fix wrong LV_PROFILER tag** 7737
- **fix(test_anim): fix stack-use-after-return** 7730
- **fix(flex): don't count item gap for leading hidden items** 7720
- **fix(drivers): generic MIPI add missing lv_display_flush_ready** 7693
- **fix(file_explorer): fix navigation when using a keypad indev** 7181
- **fix(area): lv_area_diff remove overlap** 7696
- **fix(indev.rst): handle lv_obj_is_focused no longer exists.** 7711
- **fix(sysmon): disable all performance banners from screen with serial redirect** 7593
- **fix(draw_sw_mask): add ASSERT_MALLOC check** 7692
- **fix(memcpy): ensure volatile qualifier for destination pointer in lv_…** 7573
- **fix: warn user about deprecated LV_DEFAULT_DRIVE_LETTER** 7620
- **fix(chart): fix last point not drawn in scatter chart** 7665
- **fix(canvas): initialize layer before return** 7677
- **fix(indev): skip press event on new object release** 7612
- **fix(xml): Fixed small typo** 7684
- **fix(docs): clarify obj hierarchy after `lv_menu_page_create()`** 7604
- **fix(refr): lv_obj_invalidate_area invalidates whole obj** 7598
- **fix(gif): fix bounds check** 7675
- **fix(docs): fix minor issues with alif.rst** 7676
- **fix(chart): avoid divide by zero** 7609
- **fix(docs): several minor doc fixes** 7652
- **fix(cmake): installation with custom LV_CONF_PATH** 7624
- **fix(fs_win32): fix inconsistency using LV_FS_WIN32_PATH...** 7608
- **fix(layout): always recalculate the layout if HIDDEN changes** 7607
- **fix(draw_vector): alleviate the loss of precision in obtaining path bound** 7635
- **fix(refr): lv_refr_get_top_obj not check style opa** 7643
- **fix(vg_lite_math): initialize all variables to avoid error of uninitialized variables on some compilers.** 7628
- **fix(nuttx_image_cache): fix incorrect code order** 7644
- **fix(span): fix align text to center and right layout issues.** 7615
- **fix(vg_lite): remove pattern color fill** 7613
- **fix(nxp): allow vglite build when LV_USE_DRAW_SW is disabled** 7596
- **fix(wayland): use premultiplied alpha for transparent backgrounds #7543** 7580
- **fix(os): add support for thread names** 7579
- **fix(menu): add missing LV_ASSERT_OBJ** 7605
- **fix(dave2d): handle LV_COLOR_FORMAT_RGB888 correctly** 7594
- **fix(scroll): scrolling animation and finger dragging conflict during continuous sliding, causing flickering** 7522
- **fix(pxp,vglite): extend base structure to create vglite and pxp draw units** 7578

- **fix(label): add assignment to uninitialized variable** 7587
- **fix(xml): return if XMLs can't be saved** 7588
- **fix(sysmon): fix heap memory overflow** 7576
- **fix(scroll): handle scroll chain with snapping too** 7491
- **fix(widgets): fix an incorrect text replacement** 7548
- **fix(label): limit self size by max_height style** 7542
- **fix(widget_chart): add assignment to uninitialized variable** 7561
- **fix(label): fix updating scrolling label text** 7533
- **fix(style): fix conditional jump or move depends on uninitialised value** 7571
- **fix(wayland): delete the allocated display and buffers** 7572
- **fix(docs): clarify how to pass color index...** 7554
- **fix(lv_canvas.c): fix use of uninitialized variable** 7556
- **fix(draw_buf): flush D-Cache after clear buffer** 7550
- **fix(circle_buff): replace a none ASCII character** 7538
- **fix(lv_freetype): clean up includes** 7524
- **fix(tree): add NULL check** 7526
- **fix(anim): remove the redefinition of lv_anim_set_time** 7500
- **fix(roller): Adjust the position of the lv_roller_set_str function in lv_roller. c and lv_roller. h** 7506
- **fix(font): fix get glyph id bug.** 7404
- **fix(ap): fix ap unicode 0622+0644** 7482
- **fix(sysmon): fix crash when enable LV_USE_PERF_MONITOR and nuttx backend** 7483
- **fix(event): remove preprocess flag when get name** 7468
- **fix(table): mark clicks as 'Invalid' when clicking on empty spaces** 7153
- **fix(examples): fix infinite scroll crash** 7459
- **fix(draw_sw_img): avoid divide by zero** 7447
- **fix(vg_lite): fix path memory reallocation error** 7466
- **fix(dma2d): Add return value to DMA2d conversion functions** 7456
- **fix(draw_sw_arc): add ASSERT_MALLOC check** 7448
- **fix(draw_rect): refactor to insure header is initialized** 7446
- **fix(draw_sw_triangle): add null check** 7449
- **fix(vg_lite): remove unnecessary grad image checks** 7443
- **fix(vg_lite): fix corner case of bar drawing** 7441
- **fix(osal/pthread): add missing pthread_attr_destroy call** 7434
- **fix(vg_lite): fix stroke crash when update failed** 7399
- **fix(docs): adjust colors to address accessibility issues** 7409
- **fix(cmake): fix msvc building error** 7401
- **fix(ebike-demo): make it compile with Arduino** 7397
- **fix(test_calendar.c): fix function name change from PR #7340** 7375
- **fix(draw_sw_fill): add NULL checks on grad** 7355

- **fix(arduino): allow including lvgl_private.h even if the examples and demos are in the src folder** 7366
- **fix(draw_sw): wrong image buffer calculation** 7387
- **fix(scripts): fix image conversion was scrambling palette data** 7367
- **fix(scale): fix angle calculation error** 7362
- **fix(draw_sw_box_shadow): add ASSERT_MALLOC check** 7344
- **fix(docs): add missing quotes** 7359
- **fix(theme_default): fix omitted style for selected text in `lv_textarea`** 7322
- **fix(draw): fix minor maintenance issue in lv_draw_label.c** 7296
- **fix(imagebutton): warn if middle image is not set** 7224
- **fix(init): adjust the freetype initial order** 7363
- **fix(indev): prevent division by zero** 7354
- **fix(nxp-vglite): add missing '{'** 7365
- **fix(file_explorer): quick access prototype outside guard** 7356
- **fix(lv_conf): get rid of the LV_CONF path building macros** 7335
- **fix(event): record rendering done moment after rendering finished** 7228
- **fix(draw_sw): in lv_draw_sw_rotate enable ARGB8888 functions for XRGB8888 too** 7185
- **fix(obj): use LV_ASSERT_NULL if LV_ASSERT_OBJ not enabled** 7339
- **fix(docs): stray space in style_api_gen.py and generated doc...** 7309
- **fix(issue): fix the platform input box not displaying fully** 7316
- **fix(style): missing (void*) cast in macro LV_STYLE_CONST_INIT** 7304
- **fix(flex): make min-width, grow, and wrap work together** 7168
- **fix(dave2d): required when using LVGL with LV_COLOR_DEPTH 32** 7323
- **fix(dave2d): fix implicit function definitions** 7320
- **fix(ffmpeg): add native filesystem API support for FFmpeg image decoder** 7253
- **fix(drivers/x11): fix unknown typename lv_img_dsc_t** 7294
- **fix(ebike): add guards to the ebike demo translations** 7319
- **fix(vg_lite): fix vector rendering missing image opa processing** 7293
- **fix(script): add ending for raw loader of ThorVG** 7186
- **fix(draw): fix incorrect clip area computation when clearing transparent framebuffers** 7269
- **fix(file_exploer): fix path issues when returning to open other folders after opening a certain folder** 7258
- **fix(demo): resolve include when repository name is not 'lvgl'** 7261
- **fix(file_explorer): table cell use after free** 7239
- **fix(roller): fix typo in scaling; prevent division by zero** 7263
- **fix(init): remove double call to lv_draw_sw_deinit** 7266
- **fix(indev): don't reset all indevs if an object is set disabled** 7216
- **fix(obj): fix crash with LV_SIZE_CONTENT parent and % positioned child** 7041
- **fix(draw): fix bug introduced by PR #6638** 7264
- **fix(sw_blend): add null pointer check for mask_area** 7251

- **fix(lv_draw_sw_line): fix lv_draw_line function causes a crash.** 7248
- **fix(buttonmatrix): add check for indev POINTER | BUTTON type** 7125
- **fix(freertos): compilation warning in non-IDF environments** 7221
- **fix(event) remove redundant last check** 7227
- **fix(cmake): fix installation with cmake --install** 7161
- **fix(chart): fix divide-by-zero and cursor error** 7211
- **fix(windows): Update document for Windows backend to clarify some features** 7197
- **fix(docs): fix bad highlight color** 7199
- **fix(chart): `lv_malloc()` was allocating double size needed** 7200
- **fix(refr): avoid division by zero** 7205
- **fix(buttonmatrix): fix 1-off error in `lv_buttonmatrix_get_button_text()`** 7193
- **fix(scale): correct scale_set_arc_properties function** 7113
- **fix(anim): fix user after free if the anim. is delete in the exec_cb** 7173
- **fix(cmake): generate lvgl.pc in CMAKE_CURRENT_BINARY_DIR** 7127
- **fix(demo): fix ebike demo header file inclusion issue** 7133
- **fix(table): fixed NULL pointer reference** 7042
- **fix(docs): trailing underscore looked like hyperlink to Sphinx** 7172
- **fix(docs): link in /lvgl/README.md to supported display types** 7167
- **fix(docs): fix broken links in ./lvgl/README.md** 7146
- **fix(file_explorer): fix navigation handling** 7124
- **fix(neon): use conventional macro syntax** 6887
- **fix(docs): fix incorrect title and filename change for vg_lite.rst** 7148
- **fix(draw): add assertion checks for malloc return values** 7149
- **fix(script): do not add pad if 'araw_data_len' can divide 'bblk_size' evenly.** 7109
- **fix(tile) set minimum tiles to 1 if there is no SW render unit** 7130
- **fix(ebike): make it work without lottie too** 7135
- **fix(dma2d): add missing include for descriptor structs** 7122
- **fix(vg_lite): fix linear gradient matrix error** 7110
- **fix(vg_lite): fix arc drawing boundary case drawing error** 7107
- **fix(bin_decoder): fix bug in handling premultiplied alpha flag** 7106
- **fix(tests): fix vg-lite buf address not aligned** 7049
- **fix(gdb): fix style prop getting** 7088
- **fix(obj_scroll): include handle throwing animation in scrolling judment** 7011
- **fix(tests): fix svg test assert and add missing ref images** 7079
- **fix(examples): fix font manager example build error in docs** 7085
- **fix(docs/README.md): reworked to handle several things** 6992
- **fix(codespace): use lv_conf.defaults to generate lv_conf.h** 7075
- **fix(vg_lite): fix thorvg 32bit rendering test coordinate calculation overflow** 7052
- **fix(independent_heap): add independent heap enable option** 6953

- **fix(anim_timeline): run animations which do not have an exec cb** 7043
- **fix(vg_lite): fix path bonding box coordinate overflow** 7037
- **fix(vg_lite_tvg): fix shape_set_stroke calling order error** 7039
- **fix(dave2d): fix dave2d private include issue** 7016
- **fix(anim): compensate over time** 6989
- **fix(docs): fix error blocking API-doc generation under Windows** 6990
- **fix(scale): horizontal scale sections not visible** 6982
- **fix(vg_lite_tvg): fix int32 type mismatch** 7022
- **fix(png): move png_image_free to just before return NULL** 7020
- **fix(png): fix mem leak in libpng decoder** 6952
- **fix(drivers): add missing includes** 6905
- **fix(benchmark): improve responsiveness and use the LVGL logo instead of the cogwheel** 6980
- **fix(drivers): calculate stride align correctly** 6976
- **fix(rt-thread): fix compile error** 6938
- **fix(grid): fix naming conflicts** 6963
- **fix(thorvg): fix clipped clippers** 6956
- **fix(docs): restore missing on-line examples** 6927
- **fix(docs): fix most sphinx warnings** 6916
- **fix(layout): calculate content width using x alignment** 6948
- **fix(style): remove transitions when a local style property is set** 6941
- **fix(docs): eliminate 2 types of sphinx warnings:** 6928
- **fix: eliminate misc sphinx warnings...** 6929
- **fix(calendar): fix lv_calendar_gregorian_to_chinese compile error** 6894
- **fix(fs): add lv_fs_dir_t to lv_fs.h** 6925
- **fix(indev): fix hovering disabled obj resets indev** 6855
- **fix(gif): added bounds check in gif decoder** 6863
- **fix(freertos): sync signal from isr fixed** 6793
- **fix(freertos): use xSemaphoreTakeRecursive** 6803
- **fix(dropdown): automatically center dropdown content** 6881
- **fix(draw): fix sw compile error when disable LV_DRAW_SW_COMPLEX** 6895
- **fix(textarea): fix placeholder text cannot be centered (#6879)** 6917
- **fix(libinput): private headers** 6869
- **fix(color): add missing ARGB8565 alpha check** 6891
- **fix(Kconfig): Fix non existent LV_STDLIB_BUILTIN** 6851
- **fix(display): remove the unused sw_rotate field** 6866
- **fix(image): lv_image_set_inner_align() behaviour with LV_IMAGE_ALIGN_STRETCH** 6864
- **fix(bar): fix bit overflow** 6841
- **fix(indev): don't wait until release when a new object is found** 6831
- **fix(cmake): fix install** 6787

- **fix(docbuild): update style doc to reflect Doxygen needs** 6705
- **fix(vg_lite): fix image transform clipping area error** 6810
- **fix(glfw/opengles): fix buf_size calculation error** 6830
- **fix(roller): fix stringop overflow** 6826
- **fix(perf): perf monitor FPS** 6798
- **fix(micropython): missing bidi private header feature guard** 6801
- **fix(draw): fix artifact when rotating ARGB8888 images** 6794
- **fix(sdl): check against NULL before using the driver data of a display** 6799
- **fix(assets): add missing strides** 6790
- **fix(arc): ignore hits that are outside drawn background arc** 6753
- **fix(vg_lite): fixed clip_radius image cropping error** 6780
- **fix(vg_lite/vector): convert gradient matrix to global matrix** 6577
- **fix(spangroup): fix height calculation error** 6775
- **fix(buttonmatrix): use const arrays** 6765
- **fix(ime): fix ime crash when input_char is too long** 6767
- **fix(script): follow lv_conf_template.h changes in generate_lv_conf.py** 6769
- **fix(vg_lite): select blend mode based on premultiplication** 6766
- **fix(docbuild): reformat comments for Doxygen in `lv_conf_template.h`** 6673
- **fix(draw): cast color_format in LV_DRAW_BUF_INIT_STATIC** 6729
- **fix(sdl): nested comment is not allowed** 6748
- **fix(ime_pinyin): fix letter count wrong when using some dictionary** 6752
- **fix(anim): use correct variable `repeat_cnt`** 6757
- **fix(xml): add missing style properties** d2ed41d
- **fix(font): impore static bitmap handling with stride** ccbbfcc
- **fix(vglite) upstream comments** a2b687c
- **fix(label): use LV_LABEL_LONG_MODE_* instead of LV_LABEL_LONG_*** db11e7b
- **fix(vglite) do not apply scissor cut if border is complete** 29e3da2
- **fix(refr): if tile_cnt == 1 don't create new layers.** d5b02fe
- **fix(sdl): fix caching with absolute coordinates** d469d65
- **fix(xml): use opa type for opacity reapted style props** 21e3350
- **fix(nxp) coverity issues** a107090
- **fix(xml): handle nested 'extends' handling with components** 1964dfe
- **fix(xml): minor fixes of typos and cleanups** c0a7eff
- **fix(vglite) add VGLITE_CHECK_ERROR for all vg_lite functions** 69f7599
- **fix(label): improve the position keeping of updated scrolling texts** aff9ad3
- **fix(xml): fix xml property propagation issues** 5652914
- **fix(chart): fix saving the address of a local variable in lv_example_chart_3()** 37ea110
- **fix(xml_style): add min/max_width/height and recolor** 4518d34
- **fix(event): allow using components without registering events** 22e0ebb

- **fix(vglite) Do not dispatch any task if draw unit is busy with "wait_for_finish".** b89958a
- **fix(vglite) update triangle and line dsc to align with upstream changes** 0a0fad4
- **fix(vglite) add checks for alignment, stride and static bitmap** 5564b5b
- **fix(xml): handle not existing subjects gracefully** 0b44606
- **fix(NemaGFX): arc angles** 6492096
- **fix(xml): fix typos in lv_bar.xml and lv_slider.xml** 59bc690
- **fix(draw): do not dispatch if no unit took the task** 3c07bd9
- **fix(font): add the missing flush cache operation** dcf934d
- **fix(xml): make the XML parser work on MCUs too** 2e7d17f
- **fix(nuttx_fbdev): fix memory buffer handling** b662802
- **fix(xml): fix enum type for lv_obj bind_state_if** 859b2e8
- **fix(chart): add more error handling in XML** 7256ba0
- **fix(xml): add missing align and layout values** fc4f8fa
- **fix(nxp) solve all build warnings** be9b6a2
- **fix(xml): support <view .../>** 1d71728
- **fix(ffmpeg): clear the first image data** eecebd2
- **fix(anim): make sure the resumed animtion doesn't skip the first update** eeb1fef
- **fix(vglite) ARGB8565 color format support** 284e7a4
- **fix(benchmark): better handle font dependencies** 0bbded7
- **fix(nuttx/cache): add invalidate_cache and flush_cache into image_cache_draw_buf_handlers** ea538de
- **fix(pxp) disable pxp draw tile image** d6850a8
- **fix(vglite) initialize the paths empty** 890cec7
- **fix(observer): fix 2 warnings emitted by VS C compiler...** bae25ad
- **fix(vglite) run without DRAW_THREAD** 5426204
- **fix(xml): fix roller's XML to reflect better on the C API** b27fe9d
- **fix(observer.c): fix inconsistency of whitespace around conditional directives** c28c7e8
- **fix(scale): fix typo in XML** ea18026
- **fix(xml): chart and slider minor fixes** ed7c3be
- **fix(xml): fix widget xmls** ecfe334
- **fix(vglite) fix stride calculation** 9e5039c
- **fix(vglite) skip VG_LITE_PATTERN_REPEAT tasks on GC255** 655f744
- **fix(pxp) draw_img src_area parameter** 33de407
- **fix(demo): Add missing Kconfig symbols for ebike and high res** 7d088e2
- **fix(sdl): make it work with canvas rendering too** 5d9064c
- **fix(draw): fix not reallocating draw_buf-s for glyphs** 2d0f6be
- **fix(vglite) do not draw tiled images when using blit split** 9e9d821
- **fix(vglite) draw_pattern starts the GPU after setting the command buffer** a23bdbd
- **fix(sw_blend) make sure mask_stride is initialized** 86d4385
- **fix(core):fix overflow_const issue** 1dc4096

- **fix(vglite) update get_glyph function for static bitmaps** 18cdb6a
- **fix(obj_tree): support NULL parent in lv_obj_find_by_name** 2e5931b
- **fix(test): print n-s in logs correctly** f317067
- **fix(span): fix the type of the span element** 9b061e8
- **fix(vglite) vglite draw thread stack size increase for gcFEATURE_VG_SIMPLIFIED_BEZIER** b67bfee
- **fix(ebike): fix include path** 255352b
- **fix(draw): init mask_stride when sw draw triangle** 15db6f0
- **fix(table): refresh cell size after ctrl change** 47f3c65
- **fix(obj_style): fix transition anim of style_recolor** f63b275
- **fix(style): skip figma_node_id attribute** 4b23bae
- **fix(ffmpeg): free packet** 3605c32
- **fix(observer): protect against possible negative arg value** 43b3234
- **fix(font): allow non-constant LV_FONT_DEFAULT again (fixes #7788) (#8059)** 9f090d2
- **feat(obj): add obj name support** 626d6b5
- **fix(observer): fix bug in `lv_obj_bind_checked()` function** 02c3d7c
- **fix(event): consider NULL as a wildcard for callback arg in lv_obj_remove_event_cb_with_user_data()** 0ae888b
- **fix: add +1 but for enum bitfields to avoid overflow to negative** 253981a
- **fix(sdl): add static_text flag to to lv_draw_label_dsc_t** 7808070

13.1.6 Esempi

- **example(keyboard): add draw event example to customize each button** 8172
- **example(freetype): fix docs build error** 8001
- **example(checkbox): Correct checkbox enable comment** 7763

13.1.7 Docs

- **docs(coordinates.rst): fix erroneous function...** 8283
- **docs(Renesas): add docs for Renesas RZ/A3M** 8285
- **docs(XML): proofread XML docs (batch 2 of 2)** 8185
- **docs(NuttX): Add more NuttX docs** 8235
- **docs(ft81x.rst): move under ./docs/src/ directory** 8231
- **docs(Renesas): make "Supported Boards" table cells narrower** 8214
- **docs(misc fixes): fix misc items (details below)...** 8186
- **docs(Renesas): Renesas G2UL docs change video link** 8183
- **docs(XML): proofread XML docs (batch 1 of 2)** 8184
- **docs: fix Doxygen parsing of attribute prefixes** 8179
- **docs(doxygen): exclude deprecated CJK fonts for Doxygen.** 8141
- **docs(table css): fix table formatting...** 8196
- **docs(stm32): Fix errors in FreeRTOS Example Code** 8163
- **docs(conf.py): fix URL path to viewing doc source on GitHub** 8167

- **docs(Renesas): Reorganize Renesas docs** 8153
- **docs(3dtexture.rst): get this doc into TOC and proofread** 8143
- **docs(viewe): Update docs** 8103
- **docs(gdb_plugin): add missing info draw_unit command** 8078
- **docs: add missing pcpp dependency** 8069
- **docs(introduction.rst): fix bug in API links...** 8049
- **docs(observer.rst): remove now-unnecessary warnings** 8048
- **docs(build.py): enhance build-output readability** 8028
- **docs(doc-gen): eliminate several breathe errors** 8036
- **docs(misc): misc clean-up...** 8016
- **docs(draw_layers): combine draw-layers topics...** 8034
- **docs(doc-gen): eliminate all non-API-page errors** 8035
- **docs(sdl): Add SDL docs** 8023
- **docs(draw.rst): restructure and proofread from PR #7241** 7976
- **docs(timer): document LV_NO_TIMER_READY** 7920
- **docs(README_zh.md): add missing examples section** 7897
- **docs: remove non-ASCII bytes -- batch 1** 7974
- **docs: fix erroneous api links** 7967
- **docs(getting_started.rst): fix typos (replaces PR #7829)** 7970
- **docs: fix previously-broken api links** 7971
- **docs(readme): fix broken links** 7961
- **docs(doc_builder.py): overhaul, remove bugs, clean up, modularize** 7913
- **docs(test.rst): proofread test doc, minor edits** 7906
- **docs(viewe): add Viewe docs** 7916
- **docs(test): fix the location of test.rst** 7900
- **docs: fix doc-build failure** 7879
- **docs(restructure): restructure TOC and contents for ease of use** 7847
- **docs(introduction): update active state of v8.3** 7867
- **docs(boards): proofread batch 21** 7769
- **docs: convert 4 `.rst` files back to `.md` files since not part of doc-gen** 7839
- **docs(misc): pre-re-org wrap-up...** 7804
- **docs(bindings): proofread batch 20** 7764
- **docs(xml): fix Sphinx errors in XML-component docs** 7820
- **docs(libs): proofread docs batch 19** 7709
- **docs(intro): fix typo** 7812
- **docs(libs): proofread docs batch18** 7688
- **docs(roller): fix roller API-docs spelling error** 7735
- **docs(xml): fix not using the docs** 7785
- **docs(event): add warning not to modify Widgets during draw events.** 7685

- **docs(licenses): add the missing license files** 7668
- **docs(fbdev): add information about resolution issue** 7025
- **docs(Alif): add docs for Alif chip vendor** 7622
- **docs: general typo fix on the NXP related docs.** 7626
- **docs(Riverdi): Add Riverdi board docs** 7629
- **docs(fs): pin diagram dependency version** 7453
- **docs: clarify what C functions are used in storage operations** 7627
- **docs(timer): fix usage of timer user data** 7575
- **docs(arduino): add Arduino Giga Display Shield reference** 7584
- **docs(layer): fix layer rst function error** 7541
- **docs(label): add documentation and example for text recolor** 7460
- **docs(intro): fix typos** 7485
- **docs(details): fix typos** 7484
- **docs(profiler): remove tick unit** 7433
- **docs(disp): add rotation examples** 7314
- **docs: chinese translation link only on the homepage** 7370
- **docs(buildroot): add guide to build and embed lvgl on br image** 7084
- **docs(font): add a typesetting section** 7351
- **docs: update LV_OBJ_FLAG_OVERFLOW_VISIBLE usage** 7101
- **docs: update api_json.rst** 7318
- **docs(display): correct spelling** 7225
- **docs: fix typo in Introduction: "similator" to "simulator"** 7162
- **docs(driver): fix link to example code in gen_mipi.rst** 7128
- **docs(font_manager): add documentation and test cases** 7058
- **docs(scale): fix rst** 7045
- **docs(license): add font license files** 6971
- **docs(yocto): add guide to integrate lvgl recipe in Yocto** 7024
- **docs: fix broken links** 6910
- **docs: update macro LV_TA_CURSOR_LAST => LV_TEXTAREA_CURSOR_LAST** 6995
- **docs(display): mention how to manipulate the invalidated area** 6836
- **docs(event): minor naming fixes** 6939
- **docs: fix typo in introduction** 6871
- **docs(os): fix formatting** 6829
- **docs(arm): add to index and update Arm2D docs** 6768
- **docs(xml): reorganize and extend the XML docs** ad8a9d9
- **docs(observer): proofread/edit/clarify `observer.rst`** efcdff7
- **docs(observer): re-work API (and some internal) documentation** a0e37a3
- **docs(index): sort the index links alphabetically** 032c2c3
- **docs(iter): add docstrings for lv_iter_t** 189102d

- **docs(xml): document events** 4dfdd18
- **docs(observer.rst): break up so internal TOC is...** 0944979
- **docs(API): make API TOCs easier to use** b5dc041
- **docs(ffmpeg): images always loaded with lvgl fs** 850426b
- **docs(index): complete the missing link in the index** 4d929f3

13.1.8 CI and tests

- **test(screenshot): use lodepng instead of libpng** 8275
- **ci: build examples with c++ compiler** 8261
- **ci: move docs build config to main repo** 8137
- **ci(release_updater): Add bi-weekly schedule. Only update master branches on schedule** 8074
- **ci(docs): remove concurrency clause** 8102
- **ci(docs): fix commit ref when workflow is triggered by master push** 8100
- **ci: build docs on pr** 8089
- **ci(micropython): fix MicroPython github build action** 7945
- **ci(Windows): vcpkg url patch** 7948
- **ci(ubuntu): ci use ubuntu24.04** 6241
- **ci: add 8bit build option** 7574
- **ci: fix port release branch updater git config command** 7722
- **ci: Port release updater update master as well** 7650
- **ci: cancel previous workflow run on new commit** 7619
- **ci: add port release updater** 7590
- **ci(esp): add esp32s3 build workflow** 7242
- **ci(ref_imgs): make the CI fail if there not all ref. images are added** 7086
- **ci(Kconfig): error on leading spaces (should be tabs)** 92d424e
- **ci(ccpp): add --auto-clean flag to lower disk space usage** 970ca51
- **ci: fix a reference image** 1506779

13.1.9 Others

- **chore(cmsis-pack): update cmsis-pack for v9.3.0 release** 8113
- **revert(dropdown): add lv_anim_enable_t parameter to lv_dropddown_set_selected (#7310)** 8304
- **chore: use keyword arguments in regex calls to avoid warnings** 8288
- **chore: remove extraneous file from repository** 8284
- **build(makefile): add class, instance and xml** 8233
- **build: restructure cmake** 8210
- **chore(draw/sw): minor cleanup** 8083
- **refactor(cache): rename and restructure cache-related files** 8148
- **chore(profiler): add built-in profiler default enable config** 8120
- **doc(freetype) add ftstroke.c to example Makefile** 8114
- **Update basics.rst to fix wrong variable name** 8112

- **arch(font_manager): add multiple font backend support** 8038
- **build: remove pcpp dependency** 8090
- **chore(uefi): fix typos** 8094
- **arch(demos): move new demos to lv_demos** 8022
- **build(custom): fix include path of lv_conf.cmake** 8052
- **arch(cmake): add native Kconfig support to cmake** 7934
- **Bugfix and improvement for the logging** 8025
- **chore(.gitignore): move tests directory ignores to ./tests/.gitignore** 7995
- **chore(bin_decoder): add check for image source type in `decode_compressed`** 7980
- **refact(vg_lite): refactoring vector rendering** 7928
- **chore(Kconfig): set vg_lite stroke default cache size to 8** 7939
- **chore(deps): bump JamesIves/github-pages-deploy-action from 4.7.2 to 4.7.3** 7873
- **chore(deps): bump espressif/upload-components-ci-action from 1 to 2** 7874
- **Dont's squash - gradient updates** 7646
- **chore(lv_conf_template): replace tab with space** 7736
- **chore(obj): make lv_obj_get_scroll_{left,right,top,bottom}() const** 7674
- **Feat(docs): proofread docs batch17** 7680
- **refactor(draw): use task instead of draw_unit as argument to draw functions** 7240
- **chore(draw): fix unused function warning** 7611
- **chore(deps): bump JamesIves/github-pages-deploy-action from 4.7.1 to 4.7.2** 7534
- **chore(deps): bump carlosperate/arm-none-eabi-gcc-action from 1.9.1 to 1.10.0** 7535
- **chore: fix typos in libs** 7516
- **chore(pr-template): add 'Fixes' to the template** 7462
- **chore: fix typos in lv_draw_label.h** 7511
- **chore: fix typos in lv_display.h** 7512
- **chore: fix typos in lv_text.c** 7513
- **chore: fix typos in others** 7514
- **chore: fix typos in drivers** 7517
- **chore: fix typos in some header files** 7518
- **chore: fix typos in nema_gfx** 7519
- **chore: fix typos in core** 7510
- **chore(tests): assert array size with LV_ARRAY_DEFAULT_CAPACITY** 7497
- **chore(demos): fix spelling error in demos** 7479
- **chore(indev_gesture): add missing config and fix warning** 7461
- **chore(deps): bump JamesIves/github-pages-deploy-action from 4.6.8 to 4.7.1** 7392
- **chore(display.rst): fix typo** 7376
- **chore(example_tabview): fix typo** 7377
- **chore: removes BOM** 6961

- **chore(draw_sw): add warning if the required color format is not enabled during image transformations** 7315
- **chore(nema): fix build warning** 7301
- **refactor(draw): extract function to improve readability** 7229
- **chore(api): prevent API leaks (cache and thread)** 7220
- **Feature/nema updates** 7245
- **chore(NemaGFX): add missing header, correct the docs, provided HAL is optional** 7174
- **chore(deps): bump arduino/arduino-lint-action from 1 to 2** 7209
- **chore(deps): bump actions/upload-artifact from 3 to 4** 7210
- **chore(cmsis-pack): ensure the v9.2.2 cmsis-pack is visible on keil.arm.com** 7201
- **chore(fs): treat '/' as valid driver letter** 6986
- **chore(obj_scroll): improve function documentation** 7150
- **chore(svg): LV_USE_SVG: document and enforce dependency on LV_USE_VECTOR_GRAPHIC** 7120
- **refactor(gdb): refactor gdb Debugger to be a gdb command** 7140
- **chore(vg_lite): fix path bounding_box typo** 7134
- **refactor(gdb): refactor gdb script** 7123
- **chore(demos/ebike): fixed chinese translation errors** 7102
- **chore(cmsis-pack): update cmsis pack for v9.3.0-dev** 7080
- **chore(qrcode): replaced libc includes** 7077
- **chore(docs): fix font manager example index typo** 7083
- **chore: fix minor compile warnings** 6987
- **chore(test): add missing label_recolor ref images** 7064
- **chore: remove UTF-8 BOMs** 6942
- **chore(stale): use the correct labels** 7017
- **chore(deps): bump JamesIves/github-pages-deploy-action from 4.6.3 to 4.6.8** 6984
- **chore: use the new labels in GitHub actions and docs** 6940
- **chore(lv_conf_internal): avoid LV_CONF_SKIP redefinition warning** 6960
- **chore: fix compile warnings** 6817
- **chore(widgets): add comments to the tileview and win** 6856
- **chore(vg_lite): remove unnecessary buffer checks** 6921
- **chore(Kconfig): add version info to Kconfig file to check mismatch** 6789
- **chore(array): move array typedef to lv_types.h** 6850
- **chore: add underscore prefix for names after struct and enum** 6688
- **chore(scripts): allow to run update_version.py with python3.8** 6788
- **chore(font_manager): update to v9.1 API** 6776
- **chore(deps): bump carlosperate/arm-none-eabi-gcc-action from 1.9.0 to 1.9.1** 6779
- **chore(tests): fix build break on macOS** 6773
- **chore: bump version to v9.3.0-dev** 6764
- **Revert "perf(draw): skip empty draw tasks (#6720)"** 6758

- **chore(NemaGFX): update library** 7232ad7
- **refactor(xml): rename ctx to scope as it better describes its meaning** 8e19bb4
- **refactor(vglite) use dynamic paths for drawing** e1ccefa
- **refactor(chart): add 'series' keyword in value setters/getters for consistency** 5f2888d
- **refactor(obj): use lv_obj_set_flag instead of lv_obj_update_flag** 3b394ac
- **refactor(vglite) use dynamic path_data for drawing** 4fab9de
- **refactor(vglite) remove vg from variable names** 90fa702
- **chore(class): use lv_ prefixed s to avoied naming conflicts with the user widgets** 6d99933
- **refactor(vglite) remove unnecessary arguments of draw functions** 696693b
- **refactor(iter): use circle_buf to simplify codes** b3bf2ab
- **refactor(objid): rename lv_obj_get_child_by_id to lv_obj_find_by_id for consistency** 56c44bf
- **chore(iter): add more safe checker and const qualifier** 23addd0
- **refactor(vglite) use dynamic gradients for drawing** fc43e07
- **chore(doc): add new debugging folder** a0d3efe
- **refactor(table): convert lv_table_add_cell_ctrl to a setter** 39c4011
- **refactor(pxp) remove unnecessary arguments of draw functions** 7a7edb5
- **revert(xml): revert the accidentally modified view.xml** 09d0d10
- **refactor(vglite) fix NON ASYNC mode** 8b56df4
- **chore(xml): minor fixes** 3523270
- **chore(anim): define anim on/off as true/false** 2d8ff35
- **chore(obj): deprecate lv_obj_find_by_id in favor of lv_obj_find_by_name** 346395d
- **chore(xml): fix warning** 868d7b5
- **chore(obj): add style_border_side XML support** c65eaf2
- **refactor(chart): add 'series' keyword in value setters/getters for consistency** e90cfb7
- **chore(tests): Enable -Wunused-function** b6adc8f

CAPITOLO 14

API

14.1 lv_api_map_v8.h

Defines

LV_DISP_ROTATION_0

LV_DISP_ROTATION_90

LV_DISP_ROTATION_180

LV_DISP_ROTATION_270

LV_DISP_RENDER_MODE_PARTIAL

LV_DISP_RENDER_MODE_DIRECT

LV_DISP_RENDER_MODE_FULL

LV_BTNMATRIX_BTN_NONE

LV_BTNMATRIX_CTRL_HIDDEN

LV_BTNMATRIX_CTRL_NO_REPEAT

LV_BTNMATRIX_CTRL_DISABLED

LV_BTNMATRIX_CTRL_CHECKABLE

LV_BTNMATRIX_CTRL_CHECKED

LV_BTNMATRIX_CTRL_CLICK_TRIG

LV_BTNMATRIX_CTRL_POPOVER

LV_BTNMATRIX_CTRL_CUSTOM_1

LV_BTNMATRIX_CTRL_CUSTOM_2

LV_IMGBTN_STATE_RELEASED

LV_IMGBTN_STATE_PRESSED

LV_IMGBTN_STATE_DISABLED

LV_IMGBTN_STATE_CHECKED_RELEASED

LV_IMGBTN_STATE_CHECKED_PRESSED

LV_IMGBTN_STATE_CHECKED_DISABLED

LV_RES_OK

LV_RES_INV

LV_INDEV_STATE_PR

LV_INDEV_STATE_REL

lv_obj_del

lv_obj_del_async

lv_obj_clear_flag

lv_obj_clear_state

lv_indev_set_disp

lv_indev_get_act

lv_scr_act

lv_disp_remove

lv_disp_set_default

lv_disp_get_default

lv_disp_get_next

lv_disp_set_rotation

lv_disp_get_hor_res

lv_disp_get_ver_res

lv_disp_get_physical_hor_res

lv_disp_get_physical_ver_res

lv_disp_get_offset_x

lv_disp_get_offset_y

lv_disp_get_rotation

lv_disp_get_dpi

lv_disp_get_antialiasing

lv_disp_flush_ready

lv_disp_flush_is_last

lv_disp_get_scr_act

lv_disp_get_scr_prev

lv_disp_load_scr

lv_scr_load

lv_scr_load_anim

lv_disp_get_layer_top

lv_disp_get_layer_sys

lv_disp_send_event

lv_disp_set_theme

lv_disp_get_theme

lv_disp_get_inactive_time

lv_disp_trig_activity

lv_disp_enable_invalidation

lv_disp_is_invalidation_enabled

lv_disp_refr_timer

lv_disp_get_refr_timer

lv_timer_del

lv_anim_del

lv_anim_del_all

lv_anim_set_ready_cb

lv_group_del

lv_txt_get_size

lv_txt_get_width

lv_img_create

lv_img_set_src

lv_img_set_offset_x

lv_img_set_offset_y

lv_img_set_angle

lv_img_set_pivot

lv_img_set_zoom

lv_img_set_antialias

lv_img_get_src

lv_img_get_offset_x

lv_img_get_offset_y

lv_img_get_angle

lv_img_get_pivot

lv_img_get_zoom

lv_img_get_antialias

lv_imgbtn_create

lv_imgbtn_set_src

lv_imgbtn_set_state

lv_imgbtn_get_src_left

lv_imgbtn_get_src_middle

lv_imgbtn_get_src_right

lv_list_set_btn_text

lv_list_get_btn_text

lv_list_add_btn

lv_btn_create

lv_btnmatrix_create

lv_btnmatrix_set_map

lv_btnmatrix_set_ctrl_map

lv_btnmatrix_set_selected_btn

lv_btnmatrix_set_btn_ctrl

lv_btnmatrix_clear_btn_ctrl

lv_btnmatrix_set_btn_ctrl_all

lv_btnmatrix_clear_btn_ctrl_all

lv_btnmatrix_set_btn_width

lv_btnmatrix_set_one_checked

lv_btnmatrix_get_map

lv_btnmatrix_get_selected_btn

lv_btnmatrix_get_btn_text

lv_btnmatrix_has_button_ctrl

lv_btnmatrix_get_one_checked

lv_tabview_get_tab_btns

lv_tabview_get_tab_act

lv_tabview_set_act

lv_tileview_get_tile_act

lv_obj_set_tile_id

lv_obj_set_tile

lv_roller_set_visible_row_cnt

lv_roller_get_option_cnt

lv_table_set_col_cnt

lv_table_set_row_cnt

lv_table_get_col_cnt

lv_table_get_row_cnt

lv_table_set_col_width

lv_table_get_col_width

lv_dropdown_get_option_cnt

lv_obj_get_child_cnt

lv_obj_get_disp

lv_obj_delete_anim_ready_cb

LV_STYLE_ANIM_TIME

LV_STYLE_IMG_OPA

LV_STYLE_IMG_RECOLOR

LV_STYLE_IMG_RECOLOR_OPA

LV_STYLE_SHADOW_OFS_X

LV_STYLE_SHADOW_OFS_Y

LV_STYLE_TRANSFORM_ANGLE

lv_obj_get_style_anim_time

lv_obj_get_style_img_opa

lv_obj_get_style_img_recolor

lv_obj_get_style_img_recolor_filtered

lv_obj_get_style_img_recolor_opa

lv_obj_get_style_shadow_ofs_x

lv_obj_get_style_shadow_ofs_y

lv_obj_get_style_transform_angle

lv_obj_get_style_bg_img_src

lv_obj_get_style_bg_img_recolor

lv_obj_get_style_bg_img_recolor_opa

lv_obj_set_style_anim_time

lv_obj_set_style_img_opa

lv_obj_set_style_img_recolor

lv_obj_set_style_img_recolor_opa

lv_obj_set_style_shadow_ofs_x

lv_obj_set_style_shadow_ofs_y

lv_obj_set_style_transform_zoom

lv_obj_set_style_transform_angle

lv_obj_set_style_bg_img_src

lv_obj_set_style_bg_img_recolor

lv_obj_set_style_bg_img_recolor_opa

lv_style_set_anim_time

lv_style_set_img_opa

lv_style_set_img_recolor

lv_style_set_img_recolor_opa

lv_style_set_shadow_ofs_x

lv_style_set_shadow_ofs_y

lv_style_set_transform_angle

lv_style_set_transform_zoom

lv_style_set_bg_img_src

lv_style_set_bg_img_recolor

lv_style_set_bg_img_recolor_opa

lv_keyboard_get_selected_btn

lv_keyboard_get_btn_text

LV_ZOOM_NONE

lv_image_decoder_built_in_open

lv_image_decoder_built_in_close

LV_IMG_DECLARE(var_name)

Use this macro to declare an image in a C file

Typedef

typedef int32_t **lv_coord_t**

typedef lv_result_t **lv_res_t**

typedef *lv_image_dsc_t* **lv_img_dsc_t**

typedef lv_display_t **lv_disp_t**

typedef *lv_display_rotation_t* **lv_disp_rotation_t**

typedef *lv_display_render_mode_t* **lv_disp_render_t**

typedef *lv_anim_completed_cb_t* **lv_anim_ready_cb_t**

typedef *lv_screen_load_anim_t* **lv_scr_load_anim_t**

typedef *lv_buttonmatrix_ctrl_t* **lv_btnmatrix_ctrl_t**

Functions

static inline uint32_t **lv_task_handler**(void)

static inline void **lv_obj_move_foreground**(lv_obj_t *obj)

Move the object to the foreground. It will look like if it was created as the last child of its parent. It also means it can cover any of the siblings.

Parametri

obj -- pointer to an object

static inline void **lv_obj_move_background**(lv_obj_t *obj)

Move the object to the background. It will look like if it was created as the first child of its parent. It also means any of the siblings can cover the object.

Parametri

obj -- pointer to an object

14.2 lv_api_map_v9_0.h

Defines

lv_image_set_align

lv_image_get_align

lv_button_bind_checked

LV_DRAW_BUF_DEFINE

_lv_utils_bsearch

lv_draw_buf_align_user

lv_draw_buf_create_user

lv_draw_buf_width_to_stride_user

lv_draw_buf_dup_user

lv_draw_buf_invalidate_cache_user(handlers, drawbuf, area)

lv_draw_buf_flush_cache_user(handlers, drawbuf, area)

lv_draw_buf_destroy_user(handlers, drawbuf)

14.3 lv_api_map_v9_1.h

Defines

_LV_EVENT_LAST

_lv_event_code_t

_lv_event_mark_deleted

lv_obj_add_event

_LV_STYLE_LAST_BUILT_IN_PROP

_LV_FLEX_REVERSE

_LV_FLEX_WRAP

_LV_FLEX_COLUMN

_lv_area_is_equal

_lv_area_is_in

_lv_area_intersect

_lv_area_is_point_on

_lv_area_join

_lv_image_buf_get_transformed_area

_lv_ll_init

_lv_ll_ins_head

_lv_ll_ins_prev

_lv_ll_ins_tail

_lv_ll_get_head

_lv_ll_get_tail

_lv_ll_get_next

_lv_ll_get_prev

_lv_ll_get_len

_lv_ll_move_before

_lv_ll_is_empty

_lv_ll_clear

_lv_ll_remove

_lv_ll_chg_list

_LV_LL_READ

_LV_LL_READ_BACK

_lv_obj_scroll_by_raw

_lv_obj_get_ext_draw_size

_lv_indev_scroll_handler

_lv_display_refr_timer

_lv_disp_refr_timer

_lv_disp_get_refr_timer

lv_obj_get_child_by_id

lv_obj_update_flag

_lv_inv_area

lv_chart_set_all_value

lv_calendar_set_showed_date

lv_chart_set_range

lv_chart_set_value_by_id

lv_chart_get_x_array

lv_chart_get_y_array

lv_chart_set_ext_x_array

lv_chart_set_ext_y_array

LV_LABEL_LONG_WRAP

LV_LABEL_LONG_DOT

LV_LABEL_LONG_SCROLL

LV_LABEL_LONG_SCROLL_CIRCULAR

LV_LABEL_LONG_CLIP

lv_anim_set_time

lv_anim_set_playback_time

lv_anim_set_playback_delay

lv_anim_set_playback_duration

lv_gradient_init_stops

lv_gradient_stop_t

lv_spangroup_new_span

lv_spangroup_refr_mode

lv_slider_set_left_value

lv_calendar_header_arrow_create

lv_calendar_header_dropdown_create

14.4 lv_api_map_v9_2.h

Defines

LV_SCR_LOAD_ANIM_NONE

LV_SCR_LOAD_ANIM_OVER_LEFT

LV_SCR_LOAD_ANIM_OVER_RIGHT

LV_SCR_LOAD_ANIM_OVER_TOP

LV_SCR_LOAD_ANIM_OVER_BOTTOM

LV_SCR_LOAD_ANIM_MOVE_LEFT

LV_SCR_LOAD_ANIM_MOVE_RIGHT

LV_SCR_LOAD_ANIM_MOVE_TOP

LV_SCR_LOAD_ANIM_MOVE_BOTTOM

LV_SCR_LOAD_ANIM_FADE_IN

LV_SCR_LOAD_ANIM_FADE_ON

LV_SCR_LOAD_ANIM_FADE_OUT

LV_SCR_LOAD_ANIM_OUT_LEFT

LV_SCR_LOAD_ANIM_OUT_RIGHT

LV_SCR_LOAD_ANIM_OUT_TOP

LV_SCR_LOAD_ANIM_OUT_BOTTOM

lv_ft81x_spi_operation

14.5 lv_conf_internal.h

GENERATED FILE, DO NOT EDIT IT!

This file ensures all defines of lv_conf.h have a default value.

Defines

LV_OS_NONE

LV_OS_PTHREAD

LV_OS_FREERTOS

LV_OS_CMSIS_RTOS2

LV_OS_RTTHREAD

LV_OS_WINDOWS

LV_OS_MQX

LV_OS_SDL2

LV_OS_CUSTOM

LV_STDLIB_BUILTIN

LV_STDLIB_CLIB

LV_STDLIB_MICROPYTHON

LV_STDLIB_RTTHREAD

LV_STDLIB_CUSTOM

LV_DRAW_SW_ASM_NONE

LV_DRAW_SW_ASM_NEON

LV_DRAW_SW_ASM_HELIUM

LV_DRAW_SW_ASM_CUSTOM

LV_NEMA_HAL_CUSTOM

LV_NEMA_HAL_STM32

LV_COLOR_DEPTH

Handle special Kconfig options. Color depth: 1 (I1), 8 (L8), 16 (RGB565), 24 (RGB888), 32 (XRGB8888)

LV_USE_STDLIB_MALLOC

Possible values

- LV_STDLIB_BUILTIN: LVGL's built in implementation
- LV_STDLIB_CLIB: Standard C functions, like malloc, strlen, etc
- LV_STDLIB_MICROPYTHON: MicroPython implementation
- LV_STDLIB_RTTHREAD: RT-Thread implementation
- LV_STDLIB_CUSTOM: Implement the functions externally

LV_USE_STDLIB_STRING

Possible values

- LV_STDLIB_BUILTIN: LVGL's built in implementation
- LV_STDLIB_CLIB: Standard C functions, like malloc, strlen, etc
- LV_STDLIB_MICROPYTHON: MicroPython implementation
- LV_STDLIB_RTTHREAD: RT-Thread implementation
- LV_STDLIB_CUSTOM: Implement the functions externally

LV_USE_STDLIB_SPRINTF

Possible values

- LV_STDLIB_BUILTIN: LVGL's built in implementation
- LV_STDLIB_CLIB: Standard C functions, like malloc, strlen, etc
- LV_STDLIB_MICROPYTHON: MicroPython implementation
- LV_STDLIB_RTTHREAD: RT-Thread implementation
- LV_STDLIB_CUSTOM: Implement the functions externally

LV_STDINT_INCLUDE

LV_STDDEF_INCLUDE

LV_STDBOOL_INCLUDE

LV_INTTYPES_INCLUDE

LV_LIMITS_INCLUDE

LV_STDARG_INCLUDE

LV_MEM_SIZE

Size of memory available for `lv_malloc()` in bytes (>= 2kB) [bytes]

LV_MEM_POOL_EXPAND_SIZE

Size of the memory expand for `lv_malloc()` in bytes

LV_MEM_ADR

Set an address for the memory pool instead of allocating it as a normal array. Can be in external SRAM too. 0: unused

LV_DEF_REFR_PERIOD

Default display refresh, input device read and animation step period. [ms]

LV_DPI_DEF

Default Dots Per Inch. Used to initialize default sizes such as widgets sized, style paddings. (Not so important, you can adjust it to modify default sizes and spaces.) [px/inch]

LV_USE_OS

Select operating system to use. Possible options:

- LV_OS_NONE
- LV_OS_PTHREAD
- LV_OS_FREERTOS
- LV_OS_CMSIS_RTOS2
- LV_OS_RTTHREAD
- LV_OS_WINDOWS
- LV_OS_MQX
- LV_OS_SDL2
- LV_OS_CUSTOM

LV_DRAW_BUF_STRIDE_ALIGN

Align stride of all layers and images to this bytes

LV_DRAW_BUF_ALIGN

Align start address of draw_buf addresses to this bytes

LV_DRAW_TRANSFORM_USE_MATRIX

Using matrix for transformations. Requirements:

- `LV_USE_MATRIX = 1`.
- Rendering engine needs to support 3x3 matrix transformations.

LV_DRAW_LAYER_SIMPLE_BUF_SIZE

The target buffer size for simple layer chunks. [bytes]

LV_DRAW_LAYER_MAX_MEMORY

No limit by default [bytes]

LV_DRAW_THREAD_STACK_SIZE

Stack size of drawing thread. NOTE: If FreeType or ThorVG is enabled, it is recommended to set it to 32KB or more. [bytes]

LV_DRAW_THREAD_PRIO

Thread priority of the drawing task. Higher values mean higher priority. Can use values from lv_thread_prio_t enum in lv_os.h: LV_THREAD_PRIO_LOWEST, LV_THREAD_PRIO_LOW, LV_THREAD_PRIO_MID, LV_THREAD_PRIO_HIGH, LV_THREAD_PRIO_HIGHEST Make sure the priority value aligns with the OS-specific priority levels. On systems with limited priority levels (e.g., FreeRTOS), a higher value can improve rendering performance but might cause other tasks to starve.

LV_USE_DRAW_SW

LV_DRAW_SW_SUPPORT_RGB565

LV_DRAW_SW_SUPPORT_RGB565_SWAPPED

LV_DRAW_SW_SUPPORT_RGB565A8

LV_DRAW_SW_SUPPORT_RGB888

LV_DRAW_SW_SUPPORT_XRGB8888

LV_DRAW_SW_SUPPORT_ARGB8888

LV_DRAW_SW_SUPPORT_ARGB8888_PREMULTIPLIED

LV_DRAW_SW_SUPPORT_L8

LV_DRAW_SW_SUPPORT_AL88

LV_DRAW_SW_SUPPORT_A8

LV_DRAW_SW_SUPPORT_I1

LV_DRAW_SW_I1_LUM_THRESHOLD

LV_DRAW_SW_DRAW_UNIT_CNT

Set number of draw units.

- > 1 requires operating system to be enabled in `LV_USE_OS`.
- > 1 means multiple threads will render the screen in parallel.

LV_USE_DRAW_ARM2D_SYNC

Use Arm-2D to accelerate software (sw) rendering.

LV_USE_NATIVE_HELIUM_ASM

Enable native helium assembly to be compiled.

LV_DRAW_SW_COMPLEX

- 0: Use a simple renderer capable of drawing only simple rectangles with gradient, images, text, and straight lines only.
- 1: Use a complex renderer capable of drawing rounded corners, shadow, skew lines, and arcs too.

LV_DRAW_SW_SHADOW_CACHE_SIZE

Allow buffering some shadow calculation. LV_DRAW_SW_SHADOW_CACHE_SIZE is the maximum shadow size to buffer, where shadow size is `shadow_width + radius`. Caching has LV_DRAW_SW_SHADOW_CACHE_SIZE^2 RAM cost.

LV_DRAW_SW_CIRCLE_CACHE_SIZE

Set number of maximally-cached circle data. The circumference of 1/4 circle are saved for anti-aliasing. `radius * 4` bytes are used per circle (the most often used radiuses are saved).

- 0: disables caching

LV_USE_DRAW_SW_ASM

LV_USE_DRAW_SW_COMPLEX_GRADIENTS

Enable drawing complex gradients in software: linear at an angle, radial or conical

LV_USE_NEMA_GFX

LV_USE_DRAW_VGLITE

Use NXP's VG-Lite GPU on iMX RTxxx platforms.

LV_USE_PXP

Use NXP's PXP on iMX RTxxx platforms.

LV_USE_DRAW_G2D

Use NXP's G2D on MPU platforms.

LV_USE_DRAW_DAVE2D

Use Renesas Dave2D on RA platforms.

LV_USE_DRAW_SDL

Draw using cached SDL textures

LV_USE_DRAW_VG_LITE

Use VG-Lite GPU.

LV_USE_DRAW_DMA2D

Accelerate blends, fills, etc. with STM32 DMA2D

LV_USE_DRAW_OPENGLES

Draw using cached OpenGLES textures

LV_USE_PPA

Draw using espressif PPA accelerator

LV_USE_DRAW_EVE

LV_USE_LOG

Enable log module

LV_USE_ASSERT_NULL

Check if the parameter is NULL. (Very fast, recommended)

LV_USE_ASSERT_MALLOC

Checks is the memory is successfully allocated or no. (Very fast, recommended)

LV_USE_ASSERT_STYLE

Check if the styles are properly initialized. (Very fast, recommended)

LV_USE_ASSERT_MEM_INTEGRITY

Check the integrity of `lv_mem` after critical operations. (Slow)

LV_USE_ASSERT_OBJ

Check the object's type and existence (e.g. not deleted). (Slow)

LV_ASSERT_HANDLER_INCLUDE

Add a custom handler when assert happens e.g. to restart MCU.

LV_ASSERT_HANDLER

Halt by default

LV_USE_REFR_DEBUG

1: Draw random colored rectangles over the redrawn areas.

LV_USE_LAYER_DEBUG

1: Draw a red overlay for ARGB layers and a green overlay for RGB layers

LV_USE_PARALLEL_DRAW_DEBUG

1: Adds the following behaviors for debugging:

- Draw overlays with different colors for each draw_unit's tasks.
- Draw index number of draw unit on white background.
- For layers, draws index number of draw unit on black background.

LV_ENABLE_GLOBAL_CUSTOM

LV_CACHE_DEF_SIZE

Default cache size in bytes. Used by image decoders such as `lv_lodepng` to keep the decoded image in memory. If size is not set to 0, the decoder will fail to decode when the cache is full. If size is 0, the cache function is not enabled and the decoded memory will be released immediately after use.

LV_IMAGE_HEADER_CACHE_DEF_CNT

Default number of image header cache entries. The cache is used to store the headers of images The main logic is like `LV_CACHE_DEF_SIZE` but for image headers.

LV_GRADIENT_MAX_STOPS

Number of stops allowed per gradient. Increase this to allow more stops. This adds (sizeof(lv_color_t) + 1) bytes per additional stop.

LV_COLOR_MIX_ROUND_OFS

Adjust color mix functions rounding. GPUs might calculate color mix (blending) differently.

- 0: round down,
- 64: round up from x.75,
- 128: round up from half,
- 192: round up from x.25,
- 254: round up

LV_OBJ_STYLE_CACHE

Add 2 x 32-bit variables to each `lv_obj_t` to speed up getting style properties

LV_USE_OBJ_ID

Add `id` field to `lv_obj_t`

LV_USE_OBJ_NAME

Enable support widget names

LV_OBJ_ID_AUTO_ASSIGN

Automatically assign an ID when obj is created

LV_USE_OBJ_ID_BUILTIN

Use builtin obj ID handler functions:

- lv_obj_assign_id: Called when a widget is created. Use a separate counter for each widget class as an ID.
- lv_obj_id_compare: Compare the ID to decide if it matches with a requested value.
- lv_obj_stringify_id: Return string-ified identifier, e.g. "button3".
- lv_obj_free_id: Does nothing, as there is no memory allocation for the ID. When disabled these functions needs to be implemented by the user.

LV_USE_OBJ_PROPERTY

Use obj property set/get API.

LV_USE_OBJ_PROPERTY_NAME

Enable property name support.

LV_USE_VG_LITE_THORVG

LV_USE_GESTURE_RECOGNITION

LV_BIG_ENDIAN_SYSTEM

For big endian systems set to 1

LV_ATTRIBUTE_MEM_ALIGN_SIZE

Define a custom attribute for `lv_tick_inc` function Define a custom attribute for `lv_timer_handler` function Define a custom attribute for `lv_display_flush_ready` function Align VG_LITE buffers on this number of bytes.

> **Nota**
>
> *vglite_src_buf_aligned()* uses this value to validate alignment of passed buffer pointers.

LV_EXPORT_CONST_INT(int_value)

Will be added where memory needs to be aligned (with -Os data might not be aligned to boundary by default). E.g. **attribute**((aligned(4))) Attribute to mark large constant arrays, for example for font bitmaps Compiler prefix for a large array declaration in RAM Place performance critical functions into a faster memory (e.g RAM) Export integer constant to binding. This macro is used with constants in the form of LV_<CONST> that should also appear on LVGL binding API such as MicroPython. The default value just prevents GCC warning

LV_USE_FLOAT

Prefix all global extern data with this Use `float` as `lv_value_precise_t`

LV_USE_MATRIX

Enable matrix support

- Requires `LV_USE_FLOAT = 1`

LV_USE_PRIVATE_API

Include `lvgl_private.h` in `lvgl.h` to access internal data and functions by default

LV_FONT_MONTSERRAT_8

LV_FONT_MONTSERRAT_10

LV_FONT_MONTSERRAT_12

LV_FONT_MONTSERRAT_14

LV_FONT_MONTSERRAT_16

LV_FONT_MONTSERRAT_18

LV_FONT_MONTSERRAT_20

LV_FONT_MONTSERRAT_22

LV_FONT_MONTSERRAT_24

LV_FONT_MONTSERRAT_26

LV_FONT_MONTSERRAT_28

LV_FONT_MONTSERRAT_30

LV_FONT_MONTSERRAT_32

LV_FONT_MONTSERRAT_34

LV_FONT_MONTSERRAT_36

LV_FONT_MONTSERRAT_38

LV_FONT_MONTSERRAT_40

LV_FONT_MONTSERRAT_42

LV_FONT_MONTSERRAT_44

LV_FONT_MONTSERRAT_46

LV_FONT_MONTSERRAT_48

LV_FONT_MONTSERRAT_28_COMPRESSED

bpp = 3

LV_FONT_DEJAVU_16_PERSIAN_HEBREW

Hebrew, Arabic, Persian letters and all their forms

LV_FONT_SOURCE_HAN_SANS_SC_14_CJK

1338 most common CJK radicals

LV_FONT_SOURCE_HAN_SANS_SC_16_CJK

1338 most common CJK radicals

LV_FONT_UNSCII_8

Pixel perfect monospaced fonts

LV_FONT_UNSCII_16

LV_FONT_CUSTOM_DECLARE

Optionally declare custom fonts here.

You can use any of these fonts as the default font too and they will be available globally. Example:

```
#define LV_FONT_CUSTOM_DECLARE   LV_FONT_DECLARE(my_font_1) LV_FONT_
↪DECLARE(my_font_2)
```

LV_FONT_DEFAULT

Always set a default font

LV_FONT_FMT_TXT_LARGE

Enable handling large font and/or fonts with a lot of characters. The limit depends on the font size, font face and bpp. A compiler error will be triggered if a font needs it.

LV_USE_FONT_COMPRESSED

Enables/disables support for compressed fonts.

LV_USE_FONT_PLACEHOLDER

Enable drawing placeholders when glyph dsc is not found.

LV_TXT_ENC

Select a character encoding for strings. Your IDE or editor should have the same character encoding.

- LV_TXT_ENC_UTF8
- LV_TXT_ENC_ASCII

LV_TXT_BREAK_CHARS

While rendering text strings, break (wrap) text on these chars.

LV_TXT_LINE_BREAK_LONG_LEN

If a word is at least this long, will break wherever "prettiest". To disable, set to a value <= 0.

LV_TXT_LINE_BREAK_LONG_PRE_MIN_LEN

Minimum number of characters in a long word to put on a line before a break. Depends on LV_TXT_LINE_BREAK_LONG_LEN.

LV_TXT_LINE_BREAK_LONG_POST_MIN_LEN

Minimum number of characters in a long word to put on a line after a break. Depends on LV_TXT_LINE_BREAK_LONG_LEN.

LV_USE_BIDI

Support bidirectional text. Allows mixing Left-to-Right and Right-to-Left text. The direction will be processed according to the Unicode Bidirectional Algorithm: https://www.w3.org/International/articles/inline-bidi-markup/uba-basics

LV_USE_ARABIC_PERSIAN_CHARS

Enable Arabic/Persian processing In these languages characters should be replaced with another form based on their position in the text

LV_TXT_COLOR_CMD

LV_WIDGETS_HAS_DEFAULT_VALUE

1: Causes these widgets to be given default values at creation time.

- lv_buttonmatrix_t: Get default maps: {"Btn1", "Btn2", "Btn3", "\n", "Btn4", "Btn5", ""}, else map not set.
- lv_checkbox_t : String label set to "Check box", else set to empty string.
- lv_dropdown_t : Options set to "Option 1", "Option 2", "Option 3", else no values are set.
- lv_roller_t : Options set to "Option 1", "Option 2", "Option 3", "Option 4", "Option 5", else no values are set.
- lv_label_t : Text set to "Text", else empty string.
- lv_arclabel_t : Text set to "Arced Text", else empty string.

LV_USE_ANIMIMG

LV_USE_ARC

LV_USE_ARCLABEL

LV_USE_BAR

LV_USE_BUTTON

LV_USE_BUTTONMATRIX

LV_USE_CALENDAR

LV_CALENDAR_WEEK_STARTS_MONDAY

LV_CALENDAR_DEFAULT_DAY_NAMES

LV_CALENDAR_DEFAULT_MONTH_NAMES

LV_USE_CALENDAR_HEADER_ARROW

LV_USE_CALENDAR_HEADER_DROPDOWN

LV_USE_CALENDAR_CHINESE

LV_USE_CANVAS

LV_USE_CHART

LV_USE_CHECKBOX

LV_USE_DROPDOWN

Requires: lv_label

LV_USE_IMAGE

Requires: lv_label

LV_USE_IMAGEBUTTON

LV_USE_KEYBOARD

LV_USE_LABEL

LV_LABEL_TEXT_SELECTION

Enable selecting text of the label

LV_LABEL_LONG_TXT_HINT

Store some extra info in labels to speed up drawing of very long text

LV_LABEL_WAIT_CHAR_COUNT

The count of wait chart

LV_USE_LED

LV_USE_LINE

LV_USE_LIST

LV_USE_LOTTIE

Requires: lv_canvas, thorvg

LV_USE_MENU

LV_USE_MSGBOX

LV_USE_ROLLER

Requires: lv_label

LV_USE_SCALE

LV_USE_SLIDER

Requires: lv_bar

LV_USE_SPAN

LV_SPAN_SNIPPET_STACK_SIZE

A line of text can contain this maximum number of span descriptors.

LV_USE_SPINBOX

LV_USE_SPINNER

LV_USE_SWITCH

LV_USE_TABLE

LV_USE_TABVIEW

LV_USE_TEXTAREA

Requires: lv_label

LV_TEXTAREA_DEF_PWD_SHOW_TIME

[ms]

LV_USE_TILEVIEW

LV_USE_WIN

LV_USE_3DTEXTURE

LV_USE_THEME_DEFAULT

A simple, impressive and very complete theme

LV_THEME_DEFAULT_DARK

0: Light mode; 1: Dark mode

LV_THEME_DEFAULT_GROW

1: Enable grow on press

LV_THEME_DEFAULT_TRANSITION_TIME

Default transition time in ms.

LV_USE_THEME_SIMPLE

A very simple theme that is a good starting point for a custom theme

LV_USE_THEME_MONO

A theme designed for monochrome displays

LV_USE_FLEX

A layout similar to Flexbox in CSS.

LV_USE_GRID

A layout similar to Grid in CSS.

LV_FS_DEFAULT_DRIVER_LETTER

Setting a default driver letter allows skipping the driver prefix in filepaths. Documentation about how to use the below driver-identifier letters can be found at https://docs.lvgl.io/master/details/main-modules/fs.html#lv-fs-identifier-letters .

LV_USE_FS_STDIO

API for fopen, fread, etc.

LV_USE_FS_POSIX

API for open, read, etc.

LV_USE_FS_WIN32

API for CreateFile, ReadFile, etc.

LV_USE_FS_FATFS

API for FATFS (needs to be added separately). Uses f_open, f_read, etc.

LV_USE_FS_MEMFS

API for memory-mapped file access.

LV_USE_FS_LITTLEFS

API for LittleFs.

LV_USE_FS_ARDUINO_ESP_LITTLEFS

API for Arduino LittleFs.

LV_USE_FS_ARDUINO_SD

API for Arduino Sd.

LV_USE_FS_UEFI

API for UEFI

LV_USE_LODEPNG

LODEPNG decoder library

LV_USE_LIBPNG

PNG decoder(libpng) library

LV_USE_BMP

BMP decoder library

LV_USE_TJPGD

JPG + split JPG decoder library. Split JPG is a custom format optimized for embedded systems.

LV_USE_LIBJPEG_TURBO

libjpeg-turbo decoder library.

- Supports complete JPEG specifications and high-performance JPEG decoding.

LV_USE_GIF

GIF decoder library

LV_BIN_DECODER_RAM_LOAD

Decode bin images to RAM

LV_USE_RLE

RLE decompress library

LV_USE_QRCODE

QR code library

LV_USE_BARCODE

Barcode code library

LV_USE_FREETYPE

FreeType library

LV_USE_TINY_TTF

Built-in TTF decoder

LV_USE_RLOTTIE

Rlottie library

LV_USE_VECTOR_GRAPHIC

Enable Vector Graphic APIs

- Requires `LV_USE_MATRIX = 1`

LV_USE_THORVG_INTERNAL

Enable ThorVG (vector graphics library) from the src/libs folder

LV_USE_THORVG_EXTERNAL

Enable ThorVG by assuming that its installed and linked to the project

LV_USE_LZ4_INTERNAL

Use lvgl built-in LZ4 lib

LV_USE_LZ4_EXTERNAL

Use external LZ4 library

LV_USE_SVG

LV_USE_SVG_ANIMATION

LV_USE_SVG_DEBUG

LV_USE_FFMPEG

FFmpeg library for image decoding and playing videos. Supports all major image formats so do not enable other image decoder with it.

LV_USE_SNAPSHOT

1: Enable API to take snapshot for object

LV_USE_SYSMON

1: Enable system monitor component

LV_USE_PROFILER

1: Enable runtime performance profiler

LV_USE_MONKEY

1: Enable Monkey test

LV_USE_GRIDNAV

1: Enable grid navigation

LV_USE_FRAGMENT

1: Enable `lv_obj` fragment logic

LV_USE_IMGFONT

1: Support using images as font in label or span widgets

LV_USE_OBSERVER

1: Enable an observer pattern implementation

LV_USE_IME_PINYIN

1: Enable Pinyin input method
- Requires: lv_keyboard

LV_USE_FILE_EXPLORER

1: Enable file explorer.
- Requires: lv_table

LV_USE_FONT_MANAGER

1: Enable Font manager

LV_USE_TEST

Enable emulated input devices, time emulation, and screenshot compares.

LV_USE_XML

Enable loading XML UIs runtime

LV_USE_TRANSLATION

1: Enable text translation support

LV_USE_COLOR_FILTER

LV_USE_SDL

Use SDL to open window on PC and handle mouse and keyboard.

LV_USE_X11

Use X11 to open window on Linux desktop and handle mouse and keyboard

LV_USE_WAYLAND

Use Wayland to open a window and handle input on Linux or BSD desktops

LV_USE_LINUX_FBDEV

Driver for /dev/fb

LV_USE_NUTTX

Use Nuttx to open window and handle touchscreen

LV_USE_LINUX_DRM

Driver for /dev/dri/card

LV_USE_TFT_ESPI

Interface for TFT_eSPI

LV_USE_EVDEV

Driver for evdev input devices

LV_USE_LIBINPUT

Driver for libinput input devices

LV_USE_ST7735

LV_USE_ST7789

LV_USE_ST7796

LV_USE_ILI9341

LV_USE_FT81X

LV_USE_GENERIC_MIPI

LV_USE_RENESAS_GLCDC

Driver for Renesas GLCD

LV_USE_ST_LTDC

Driver for ST LTDC

LV_USE_NXP_ELCDIF

Driver for NXP ELCDIF

LV_USE_WINDOWS

LVGL Windows backend

LV_USE_UEFI

LVGL UEFI backend

LV_USE_OPENGLES

Use OpenGL to open window on PC and handle mouse and keyboard

LV_USE_QNX

QNX Screen display and input drivers

LV_BUILD_EXAMPLES

Enable examples to be built with the library.

LV_BUILD_DEMOS

Build the demos

LV_USE_DEMO_WIDGETS

Show some widgets. This might be required to increase `LV_MEM_SIZE`.

LV_USE_DEMO_KEYPAD_AND_ENCODER

Demonstrate usage of encoder and keyboard.

LV_USE_DEMO_BENCHMARK

Benchmark your system

LV_USE_DEMO_RENDER

Render test for each primitive.

- Requires at least 480x272 display.

LV_USE_DEMO_STRESS

Stress test for LVGL

LV_USE_DEMO_MUSIC

Music player demo

LV_USE_DEMO_VECTOR_GRAPHIC

Vector graphic demo

LV_USE_DEMO_FLEX_LAYOUT

Flex layout demo

LV_USE_DEMO_MULTILANG

Smart-phone like multi-language demo

LV_USE_DEMO_TRANSFORM

Widget transformation demo

LV_USE_DEMO_SCROLL

Demonstrate scroll settings

LV_USE_DEMO_EBIKE

LV_USE_DEMO_HIGH_RES

High-resolution demo

LV_USE_DEMO_SMARTWATCH

LV_USE_ANIMIMAGE

LV_LOG_LEVEL

LV_LOG_TRACE_MEM

LV_LOG_TRACE_TIMER

LV_LOG_TRACE_INDEV

LV_LOG_TRACE_DISP_REFR

LV_LOG_TRACE_EVENT

LV_LOG_TRACE_OBJ_CREATE

LV_LOG_TRACE_LAYOUT

LV_LOG_TRACE_ANIM

LV_WAYLAND_USE_DMABUF

LV_WAYLAND_WINDOW_DECORATIONS

LV_WAYLAND_WL_SHELL

LV_USE_PERF_MONITOR

LV_USE_MEM_MONITOR

LV_USE_PERF_MONITOR_LOG_MODE

LV_USE_LZ4

LV_USE_THORVG

LV_FS_IS_VALID_LETTER(1)

14.6 lv_conf_kconfig.h

- * Configs that need special handling when LVGL is used with Kconfig

14.7 lv_init.h

Functions

void **lv_init**(void)

Initialize LVGL library. Should be called before any other LVGL related function.

void **lv_deinit**(void)

Deinit the 'lv' library

bool **lv_is_initialized**(void)

Returns whether the 'lv' library is currently initialized

14.8 lvgl.h

This file exists only to be compatible with Arduino's library structure

14.9 lvgl_private.h

This file exists only to be compatible with Arduino's library structure

14.10 core

14.10.1 lv_global.h

Defines

ZERO_MEM_SENTINEL

LV_GLOBAL_DEFAULT()

Typedef

typedef struct *_lv_global_t* **lv_global_t**

Variables

lv_global_t **lv_global**

struct **_lv_global_t**

Public Members

bool **inited**

bool **deinit_in_progress**

Can be used e.g. in the LV_EVENT_DELETE to deinit the drivers too

lv_ll_t **disp_ll**

lv_display_t ***disp_refresh**

lv_display_t ***disp_default**

lv_ll_t **style_trans_ll**

bool **style_refresh**

uint32_t **style_custom_table_size**

uint32_t **style_last_custom_prop_id**

uint8_t ***style_custom_prop_flag_lookup_table**

lv_ll_t **group_ll**

lv_group_t ***group_default**

lv_ll_t **indev_ll**

lv_indev_t ***indev_active**

lv_obj_t ***indev_obj_active**

uint32_t **layout_count**

lv_layout_dsc_t ***layout_list**

bool **layout_update_mutex**

uint32_t **memory_zero**

uint32_t **math_rand_seed**

lv_event_t ***event_header**

uint32_t **event_last_register_id**

lv_timer_state_t **timer_state**

lv_anim_state_t **anim_state**

lv_tick_state_t **tick_state**

lv_draw_buf_handlers_t **draw_buf_handlers**

lv_draw_buf_handlers_t **font_draw_buf_handlers**

lv_draw_buf_handlers_t **image_cache_draw_buf_handlers**

Ensure that all assigned draw buffers can be managed by image cache.

lv_ll_t **img_decoder_ll**

lv_cache_t ***img_cache**

lv_cache_t ***img_header_cache**

lv_draw_global_info_t **draw_info**

lv_ll_t **draw_sw_blend_handler_ll**

lv_draw_sw_mask_radius_circle_dsc_arr_t **sw_circle_cache**

lv_log_print_g_cb_t **custom_log_print_cb**

uint32_t **log_last_log_time**

void ***theme_simple**

void ***theme_default**

void ***theme_mono**

lv_tlsf_state_t **tlsf_state**

lv_ll_t **fsdrv_ll**

lv_fs_drv_t **stdio_fs_drv**

lv_fs_drv_t **posix_fs_drv**

lv_fs_drv_t **fatfs_fs_drv**

lv_fs_drv_t **win32_fs_drv**

lv_fs_drv_t **uefi_fs_drv**

lv_fs_drv_t **littlefs_fs_drv**

lv_fs_drv_t **arduino_esp_littlefs_fs_drv**

lv_fs_drv_t **arduino_sd_fs_drv**

struct *_lv_freetype_context_t* ***ft_context**

lv_font_fmt_rle_t **font_fmt_rle**

struct _snippet_stack ***span_snippet_stack**

lv_style_t **fe_list_button_style**

lv_sysmon_backend_data_t **sysmon_mem**

size_t **ime_cand_len**

void ***objid_array**

uint32_t **objid_count**

lv_test_state_t **test_state**

lv_ll_t **translation_packs_ll**

const char ***translation_selected_lang**

struct *_lv_nuttx_ctx_t* ***nuttx_ctx**

lv_evdev_discovery_t ***evdev_discovery**

const char ***xml_path_prefix**

lv_draw_eve_unit_t ***draw_eve_unit**

void ***user_data**

14.10.2 lv_group.h

Typedef

typedef void (***lv_group_focus_cb_t**)(lv_group_t*)

typedef void (***lv_group_edge_cb_t**)(lv_group_t*, bool)

Enums

enum **lv_key_t**

Predefined keys to control which Widget has focus via lv_group_send(group, c)

Values:

enumerator **LV_KEY_UP**

enumerator **LV_KEY_DOWN**

enumerator **LV_KEY_RIGHT**

enumerator **LV_KEY_LEFT**

enumerator **LV_KEY_ESC**

enumerator **LV_KEY_DEL**

enumerator **LV_KEY_BACKSPACE**

enumerator **LV_KEY_ENTER**

enumerator **LV_KEY_NEXT**

enumerator **LV_KEY_PREV**

enumerator **LV_KEY_HOME**

enumerator **LV_KEY_END**

enum **lv_group_refocus_policy_t**

Values:

enumerator **LV_GROUP_REFOCUS_POLICY_NEXT**

enumerator **LV_GROUP_REFOCUS_POLICY_PREV**

Functions

lv_group_t ***lv_group_create**(void)

Create new Widget group.

Ritorna

pointer to the new Widget group

void **lv_group_delete**(lv_group_t *group)

Delete group object.

Parametri

group -- pointer to a group

void **lv_group_set_default**(lv_group_t *group)

Set default group. New Widgets will be added to this group if it's enabled in their class with `add_to_def_group = true`.

Parametri

group -- pointer to a group (can be `NULL`)

lv_group_t ***lv_group_get_default**(void)

Get default group.

Ritorna

pointer to the default group

void **lv_group_add_obj**(lv_group_t *group, lv_obj_t *obj)

Add an Widget to group.

Parametri

- **group** -- pointer to a group
- **obj** -- pointer to a Widget to add

void **lv_group_swap_obj**(lv_obj_t *obj1, lv_obj_t *obj2)

Swap 2 Widgets in group. Widgets must be in the same group.

Parametri

- **obj1** -- pointer to a Widget
- **obj2** -- pointer to another Widget

void **lv_group_remove_obj**(lv_obj_t *obj)

Remove a Widget from its group.

Parametri

obj -- pointer to Widget to remove

void **lv_group_remove_all_objs**(lv_group_t *group)

Remove all Widgets from a group.

Parametri

group -- pointer to a group

void **lv_group_focus_obj**(lv_obj_t *obj)

Focus on a Widget (defocus the current).

Parametri

obj -- pointer to Widget to focus on

void **lv_group_focus_next**(lv_group_t *group)

Focus on next Widget in a group (defocus the current).

Parametri

group -- pointer to a group

void **lv_group_focus_prev**(lv_group_t *group)

Focus on previous Widget in a group (defocus the current).

Parametri

group -- pointer to a group

void **lv_group_focus_freeze**(lv_group_t *group, bool en)

Do not allow changing focus from current Widget.

Parametri

- **group** -- pointer to a group
- **en** -- true: freeze, false: release freezing (normal mode)

lv_result_t **lv_group_send_data**(lv_group_t *group, uint32_t c)

Send a control character to Widget that has focus in a group.

Parametri

- **group** -- pointer to a group
- **c** -- a character (use LV_KEY_.. to navigate)

Ritorna

result of Widget with focus in group.

void **lv_group_set_focus_cb**(lv_group_t *group, *lv_group_focus_cb_t* focus_cb)

Set a function for a group which will be called when a new Widget has focus.

Parametri

- **group** -- pointer to a group
- **focus_cb** -- the call back function or NULL if unused

void **lv_group_set_edge_cb**(lv_group_t *group, *lv_group_edge_cb_t* edge_cb)

Set a function for a group which will be called when a focus edge is reached

Parametri

- **group** -- pointer to a group
- **edge_cb** -- the call back function or NULL if unused

void **lv_group_set_refocus_policy**(lv_group_t *group, *lv_group_refocus_policy_t* policy)

Set whether the next or previous Widget in a group gets focus when Widget that has focus is deleted.

Parametri

- **group** -- pointer to a group
- **policy** -- new refocus policy enum

void **lv_group_set_editing**(lv_group_t *group, bool edit)

Manually set the current mode (edit or navigate).

Parametri

- **group** -- pointer to group
- **edit** -- true: edit mode; false: navigate mode

void **lv_group_set_wrap**(lv_group_t *group, bool en)

Set whether moving focus to next/previous Widget will allow wrapping from first->last or last->first Widget.

Parametri

- **group** -- pointer to group
- **en** -- true: wrapping enabled; false: wrapping disabled

lv_obj_t ***lv_group_get_focused**(const lv_group_t *group)

Get Widget that has focus, or NULL if there isn't one.

Parametri

group -- pointer to a group

Ritorna

pointer to Widget with focus

lv_group_focus_cb_t **lv_group_get_focus_cb**(const lv_group_t *group)

Get focus callback function of a group.

Parametri

group -- pointer to a group

Ritorna
the call back function or NULL if not set

lv_group_edge_cb_t **lv_group_get_edge_cb**(const lv_group_t *group)

Get edge callback function of a group.

Parametri
group -- pointer to a group

Ritorna
the call back function or NULL if not set

bool **lv_group_get_editing**(const lv_group_t *group)

Get current mode (edit or navigate).

Parametri
group -- pointer to group

Ritorna
true: edit mode; false: navigate mode

bool **lv_group_get_wrap**(lv_group_t *group)

Get whether moving focus to next/previous Widget will allow wrapping from first->last or last->first Widget.

Parametri
group -- pointer to group

uint32_t **lv_group_get_obj_count**(lv_group_t *group)

Get number of Widgets in group.

Parametri
group -- pointer to a group

Ritorna
number of Widgets in the group

lv_obj_t ***lv_group_get_obj_by_index**(lv_group_t *group, uint32_t index)

Get nth Widget within group.

Parametri
- **group** -- pointer to a group
- **index** -- index of Widget within the group

Ritorna
pointer to Widget

uint32_t **lv_group_get_count**(void)

Get the number of groups.

Ritorna
number of groups

lv_group_t ***lv_group_by_index**(uint32_t index)

Get a group by its index.

Parametri
index -- index of the group

Ritorna
pointer to the group

14.10.3 lv_group_private.h

Functions

void **lv_group_init**(void)

Init the group module

> **Remark**
> Internal function, do not call directly.

void **lv_group_deinit**(void)

Deinit the group module

> **Remark**
>
> Internal function, do not call directly.

struct **_lv_group_t**

#include <lv_group_private.h> Groups can be used to logically hold objects so that they can be individually focused. They are NOT for laying out objects on a screen (try layouts for that).

Public Members

lv_ll_t **obj_ll**

Linked list to store the objects in the group

lv_obj_t ****obj_focus**

The object in focus

lv_group_focus_cb_t **focus_cb**

A function to call when a new object is focused (optional)

lv_group_edge_cb_t **edge_cb**

A function to call when an edge is reached, no more focus targets are available in this direction (to allow edge feedback like a sound or a scroll bounce)

void ***user_data**

uint8_t **frozen**

1: can't focus to new object

uint8_t **editing**

1: Edit mode, 0: Navigate mode

uint8_t **refocus_policy**

1: Focus prev if focused on deletion. 0: Focus next if focused on deletion.

uint8_t **wrap**

1: Focus next/prev can wrap at end of list. 0: Focus next/prev stops at end of list.

14.10.4 lv_obj.h

Defines

LV_ASSERT_OBJ(obj_p, obj_class)

LV_TRACE_OBJ_CREATE(...)

Enums

enum **lv_obj_flag_t**

On/Off features controlling the object's behavior. OR-ed values are possible

Note: update obj flags corresponding properties below whenever add/remove flags or change bit definition of flags.

Values:

enumerator **LV_OBJ_FLAG_HIDDEN**

Make the object hidden. (Like it wasn't there at all)

enumerator **LV_OBJ_FLAG_CLICKABLE**

Make the object clickable by the input devices

enumerator **LV_OBJ_FLAG_CLICK_FOCUSABLE**

Add focused state to the object when clicked

enumerator **LV_OBJ_FLAG_CHECKABLE**

Toggle checked state when the object is clicked

enumerator **LV_OBJ_FLAG_SCROLLABLE**

Make the object scrollable

enumerator **LV_OBJ_FLAG_SCROLL_ELASTIC**

Allow scrolling inside but with slower speed

enumerator **LV_OBJ_FLAG_SCROLL_MOMENTUM**

Make the object scroll further when "thrown"

enumerator **LV_OBJ_FLAG_SCROLL_ONE**

Allow scrolling only one snappable children

enumerator **LV_OBJ_FLAG_SCROLL_CHAIN_HOR**

Allow propagating the horizontal scroll to a parent

enumerator **LV_OBJ_FLAG_SCROLL_CHAIN_VER**

Allow propagating the vertical scroll to a parent

enumerator **LV_OBJ_FLAG_SCROLL_CHAIN**

enumerator **LV_OBJ_FLAG_SCROLL_ON_FOCUS**

Automatically scroll object to make it visible when focused

enumerator **LV_OBJ_FLAG_SCROLL_WITH_ARROW**

Allow scrolling the focused object with arrow keys

enumerator **LV_OBJ_FLAG_SNAPPABLE**

If scroll snap is enabled on the parent it can snap to this object

enumerator **LV_OBJ_FLAG_PRESS_LOCK**

Keep the object pressed even if the press slid from the object

enumerator **LV_OBJ_FLAG_EVENT_BUBBLE**

Propagate the events to the parent too

enumerator **LV_OBJ_FLAG_GESTURE_BUBBLE**

Propagate the gestures to the parent

enumerator **LV_OBJ_FLAG_ADV_HITTEST**

Allow performing more accurate hit (click) test. E.g. consider rounded corners.

enumerator **LV_OBJ_FLAG_IGNORE_LAYOUT**

Make the object not positioned by the layouts

enumerator **LV_OBJ_FLAG_FLOATING**

Do not scroll the object when the parent scrolls and ignore layout

enumerator **LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS**

Send `LV_EVENT_DRAW_TASK_ADDED` events

enumerator **LV_OBJ_FLAG_OVERFLOW_VISIBLE**

Do not clip the children to the parent's ext draw size

enumerator **LV_OBJ_FLAG_EVENT_TRICKLE**

Propagate the events to the children too

enumerator **LV_OBJ_FLAG_STATE_TRICKLE**

Propagate the states to the children too

enumerator **LV_OBJ_FLAG_LAYOUT_1**

Custom flag, free to use by layouts

enumerator **LV_OBJ_FLAG_LAYOUT_2**

Custom flag, free to use by layouts

enumerator **LV_OBJ_FLAG_FLEX_IN_NEW_TRACK**

Start a new flex track on this item

enumerator **LV_OBJ_FLAG_WIDGET_1**

Custom flag, free to use by widget

enumerator **LV_OBJ_FLAG_WIDGET_2**

Custom flag, free to use by widget

enumerator **LV_OBJ_FLAG_USER_1**

Custom flag, free to use by user

enumerator **LV_OBJ_FLAG_USER_2**
Custom flag, free to use by user

enumerator **LV_OBJ_FLAG_USER_3**
Custom flag, free to use by user

enumerator **LV_OBJ_FLAG_USER_4**
Custom flag, free to use by user

enum **_lv_signed_prop_id_t**
Values:

enumerator **LV_PROPERTY_OBJ_FLAG_START**

enumerator **LV_PROPERTY_OBJ_FLAG_HIDDEN**

enumerator **LV_PROPERTY_OBJ_FLAG_CLICKABLE**

enumerator **LV_PROPERTY_OBJ_FLAG_CLICK_FOCUSABLE**

enumerator **LV_PROPERTY_OBJ_FLAG_CHECKABLE**

enumerator **LV_PROPERTY_OBJ_FLAG_SCROLLABLE**

enumerator **LV_PROPERTY_OBJ_FLAG_SCROLL_ELASTIC**

enumerator **LV_PROPERTY_OBJ_FLAG_SCROLL_MOMENTUM**

enumerator **LV_PROPERTY_OBJ_FLAG_SCROLL_ONE**

enumerator **LV_PROPERTY_OBJ_FLAG_SCROLL_CHAIN_HOR**

enumerator **LV_PROPERTY_OBJ_FLAG_SCROLL_CHAIN_VER**

enumerator **LV_PROPERTY_OBJ_FLAG_SCROLL_ON_FOCUS**

enumerator **LV_PROPERTY_OBJ_FLAG_SCROLL_WITH_ARROW**

enumerator **LV_PROPERTY_OBJ_FLAG_SNAPPABLE**

enumerator **LV_PROPERTY_OBJ_FLAG_PRESS_LOCK**

enumerator **LV_PROPERTY_OBJ_FLAG_EVENT_BUBBLE**

enumerator **LV_PROPERTY_OBJ_FLAG_GESTURE_BUBBLE**

enumerator **LV_PROPERTY_OBJ_FLAG_ADV_HITTEST**

enumerator **LV_PROPERTY_OBJ_FLAG_IGNORE_LAYOUT**

enumerator **LV_PROPERTY_OBJ_FLAG_FLOATING**

enumerator **LV_PROPERTY_OBJ_FLAG_SEND_DRAW_TASK_EVENTS**

enumerator **LV_PROPERTY_OBJ_FLAG_OVERFLOW_VISIBLE**

enumerator **LV_PROPERTY_OBJ_FLAG_EVENT_TRICKLE**

enumerator **LV_PROPERTY_OBJ_FLAG_STATE_TRICKLE**

enumerator **LV_PROPERTY_OBJ_FLAG_LAYOUT_1**

enumerator **LV_PROPERTY_OBJ_FLAG_LAYOUT_2**

enumerator **LV_PROPERTY_OBJ_FLAG_FLEX_IN_NEW_TRACK**

enumerator **LV_PROPERTY_OBJ_FLAG_WIDGET_1**

enumerator **LV_PROPERTY_OBJ_FLAG_WIDGET_2**

enumerator **LV_PROPERTY_OBJ_FLAG_USER_1**

enumerator **LV_PROPERTY_OBJ_FLAG_USER_2**

enumerator **LV_PROPERTY_OBJ_FLAG_USER_3**

enumerator **LV_PROPERTY_OBJ_FLAG_USER_4**

enumerator **LV_PROPERTY_OBJ_FLAG_END**

enumerator **LV_PROPERTY_OBJ_STATE_START**

enumerator **LV_PROPERTY_OBJ_STATE_CHECKED**

enumerator **LV_PROPERTY_OBJ_STATE_FOCUSED**

enumerator **LV_PROPERTY_OBJ_STATE_FOCUS_KEY**

enumerator **LV_PROPERTY_OBJ_STATE_EDITED**

enumerator **LV_PROPERTY_OBJ_STATE_HOVERED**

enumerator **LV_PROPERTY_OBJ_STATE_PRESSED**

enumerator **LV_PROPERTY_OBJ_STATE_SCROLLED**

enumerator **LV_PROPERTY_OBJ_STATE_DISABLED**

enumerator **LV_PROPERTY_OBJ_STATE_USER_1**

enumerator **LV_PROPERTY_OBJ_STATE_USER_2**

enumerator **LV_PROPERTY_OBJ_STATE_USER_3**

enumerator **LV_PROPERTY_OBJ_STATE_USER_4**

enumerator **LV_PROPERTY_OBJ_STATE_ANY**

enumerator **LV_PROPERTY_OBJ_STATE_END**

enumerator **LV_PROPERTY_OBJ_PARENT**

enumerator **LV_PROPERTY_OBJ_X**

enumerator **LV_PROPERTY_OBJ_Y**

enumerator **LV_PROPERTY_OBJ_W**

enumerator **LV_PROPERTY_OBJ_H**

enumerator **LV_PROPERTY_OBJ_CONTENT_WIDTH**

enumerator **LV_PROPERTY_OBJ_CONTENT_HEIGHT**

enumerator **LV_PROPERTY_OBJ_LAYOUT**

enumerator **LV_PROPERTY_OBJ_ALIGN**

enumerator **LV_PROPERTY_OBJ_SCROLLBAR_MODE**

enumerator **LV_PROPERTY_OBJ_SCROLL_DIR**

enumerator **LV_PROPERTY_OBJ_SCROLL_SNAP_X**

enumerator **LV_PROPERTY_OBJ_SCROLL_SNAP_Y**

enumerator **LV_PROPERTY_OBJ_SCROLL_X**

enumerator **LV_PROPERTY_OBJ_SCROLL_Y**

enumerator **LV_PROPERTY_OBJ_SCROLL_TOP**

enumerator **LV_PROPERTY_OBJ_SCROLL_BOTTOM**

enumerator **LV_PROPERTY_OBJ_SCROLL_LEFT**

enumerator **LV_PROPERTY_OBJ_SCROLL_RIGHT**

enumerator **LV_PROPERTY_OBJ_SCROLL_END**

enumerator **LV_PROPERTY_OBJ_EXT_DRAW_SIZE**

enumerator **LV_PROPERTY_OBJ_EVENT_COUNT**

enumerator **LV_PROPERTY_OBJ_SCREEN**

enumerator **LV_PROPERTY_OBJ_DISPLAY**

enumerator **LV_PROPERTY_OBJ_CHILD_COUNT**

enumerator **LV_PROPERTY_OBJ_INDEX**

enumerator **LV_PROPERTY_OBJ_END**

Functions

lv_obj_t ***lv_obj_create**(lv_obj_t *parent)

Create a base object (a rectangle)

Parametri

parent -- pointer to a parent object. If NULL then a screen will be created.

Ritorna

pointer to the new object

void **lv_obj_add_flag**(lv_obj_t *obj, *lv_obj_flag_t* f)

Set one or more flags

Parametri

- **obj** -- pointer to an object
- **f** -- OR-ed values from *lv_obj_flag_t* to set.

void **lv_obj_remove_flag**(lv_obj_t *obj, *lv_obj_flag_t* f)

Remove one or more flags

Parametri

- **obj** -- pointer to an object
- **f** -- OR-ed values from *lv_obj_flag_t* to clear.

void **lv_obj_set_flag**(lv_obj_t *obj, *lv_obj_flag_t* f, bool v)

Set add or remove one or more flags.

Parametri

- **obj** -- pointer to an object
- **f** -- OR-ed values from *lv_obj_flag_t* to update.

- **v** -- true: add the flags; false: remove the flags

void **lv_obj_add_state**(lv_obj_t *obj, *lv_state_t* state)

Add one or more states to the object. The other state bits will remain unchanged. If specified in the styles, transition animation will be started from the previous state to the current.

Parametri

- **obj** -- pointer to an object
- **state** -- the states to add. E.g `LV_STATE_PRESSED | LV_STATE_FOCUSED`

void **lv_obj_remove_state**(lv_obj_t *obj, *lv_state_t* state)

Remove one or more states to the object. The other state bits will remain unchanged. If specified in the styles, transition animation will be started from the previous state to the current.

Parametri

- **obj** -- pointer to an object
- **state** -- the states to add. E.g `LV_STATE_PRESSED | LV_STATE_FOCUSED`

void **lv_obj_set_state**(lv_obj_t *obj, *lv_state_t* state, bool v)

Add or remove one or more states to the object. The other state bits will remain unchanged.

Parametri

- **obj** -- pointer to an object
- **state** -- the states to add. E.g `LV_STATE_PRESSED | LV_STATE_FOCUSED`
- **v** -- true: add the states; false: remove the states

void **lv_obj_set_user_data**(lv_obj_t *obj, void *user_data)

Set the user_data field of the object

Parametri

- **obj** -- pointer to an object
- **user_data** -- pointer to the new user_data.

bool **lv_obj_has_flag**(const lv_obj_t *obj, *lv_obj_flag_t* f)

Check if a given flag or all the given flags are set on an object.

Parametri

- **obj** -- pointer to an object
- **f** -- the flag(s) to check (OR-ed values can be used)

Ritorna

true: all flags are set; false: not all flags are set

bool **lv_obj_has_flag_any**(const lv_obj_t *obj, *lv_obj_flag_t* f)

Check if a given flag or any of the flags are set on an object.

Parametri

- **obj** -- pointer to an object
- **f** -- the flag(s) to check (OR-ed values can be used)

Ritorna

true: at least one flag is set; false: none of the flags are set

lv_state_t **lv_obj_get_state**(const lv_obj_t *obj)

Get the state of an object

Parametri

obj -- pointer to an object

Ritorna

the state (OR-ed values from `lv_state_t`)

bool **lv_obj_has_state**(const lv_obj_t *obj, *lv_state_t* state)

Check if the object is in a given state or not.

Parametri

- **obj** -- pointer to an object
- **state** -- a state or combination of states to check

Ritorna
true: `obj` is in `state`; false: `obj` is not in `state`

lv_group_t *`lv_obj_get_group`(const lv_obj_t *obj)

Get the group of the object

Parametri
`obj` -- pointer to an object

Ritorna
the pointer to group of the object

void *`lv_obj_get_user_data`(lv_obj_t *obj)

Get the user_data field of the object

Parametri
`obj` -- pointer to an object

Ritorna
the pointer to the user_data of the object

void `lv_obj_allocate_spec_attr`(lv_obj_t *obj)

Allocate special data for an object if not allocated yet.

Parametri
`obj` -- pointer to an object

bool `lv_obj_check_type`(const lv_obj_t *obj, const lv_obj_class_t *class_p)

Check the type of obj.

Parametri
- **`obj`** -- pointer to an object
- **`class_p`** -- a class to check (e.g. `lv_slider_class`)

Ritorna
true: `class_p` is the `obj` class.

bool `lv_obj_has_class`(const lv_obj_t *obj, const lv_obj_class_t *class_p)

Check if any object has a given class (type). It checks the ancestor classes too.

Parametri
- **`obj`** -- pointer to an object
- **`class_p`** -- a class to check (e.g. `lv_slider_class`)

Ritorna
true: `obj` has the given class

const lv_obj_class_t *`lv_obj_get_class`(const lv_obj_t *obj)

Get the class (type) of the object

Parametri
`obj` -- pointer to an object

Ritorna
the class (type) of the object

bool `lv_obj_is_valid`(const lv_obj_t *obj)

Check if any object is still "alive".

Parametri
`obj` -- pointer to an object

Ritorna
true: valid

void `lv_obj_null_on_delete`(lv_obj_t **obj_ptr)

Utility to set an object reference to NULL when it gets deleted. The reference should be in a location that will not become invalid during the object's lifetime, i.e. static or allocated.

Parametri
`obj_ptr` -- a pointer to a pointer to an object

void `lv_obj_add_screen_load_event`(lv_obj_t *obj, *lv_event_code_t* trigger, lv_obj_t *screen, *lv_screen_load_anim_t* anim_type, uint32_t duration, uint32_t delay)

Add an event handler to a widget that will load a screen on a trigger.

Parametri

- **obj** -- pointer to widget which should load the screen
- **trigger** -- an event code, e.g. `LV_EVENT_CLICKED`
- **screen** -- the screen to load (must be a valid widget)
- **anim_type** -- element of `lv_screen_load_anim_t` the screen load animation
- **duration** -- duration of the animation in milliseconds
- **delay** -- delay before the screen load in milliseconds

void **lv_obj_add_screen_create_event**(lv_obj_t *obj, *lv_event_code_t* trigger, lv_screen_create_cb_t screen_create_cb, *lv_screen_load_anim_t* anim_type, uint32_t duration, uint32_t delay)

Add an event handler to a widget that will create a screen on a trigger. The created screen will be deleted when it's unloaded

Parametri

- **obj** -- pointer to widget which should load the screen
- **trigger** -- an event code, e.g. `LV_EVENT_CLICKED`
- **screen_create_cb** -- a callback to create the screen, e.g. `lv_obj_t * myscreen_create(void)`
- **anim_type** -- element of `lv_screen_load_anim_t` the screen load animation
- **duration** -- duration of the animation in milliseconds
- **delay** -- delay before the screen load in milliseconds

void **lv_obj_set_id**(lv_obj_t *obj, void *id)

Set an id for an object.

Parametri

- **obj** -- pointer to an object
- **id** -- the id of the object

void ***lv_obj_get_id**(const lv_obj_t *obj)

Get the id of an object.

Parametri

obj -- pointer to an object

Ritorna

the id of the object

lv_obj_t ***lv_obj_find_by_id**(const lv_obj_t *obj, const void *id)

DEPRECATED IDs are used only to print the widget trees. To find a widget use `lv_obj_find_by_name`

Get the child object by its id. It will check children and grandchildren recursively. Function `lv_obj_id_compare` is used to matched obj id with given id.

Parametri

- **obj** -- pointer to an object
- **id** -- the id of the child object

Ritorna

pointer to the child object or NULL if not found

void **lv_obj_assign_id**(const lv_obj_class_t *class_p, lv_obj_t *obj)

Assign id to object if not previously assigned. This function gets called automatically when LV_OBJ_ID_AUTO_ASSIGN is enabled.

Set `LV_USE_OBJ_ID_BUILTIN` to use the builtin method to generate object ID. Otherwise, these functions including `lv_obj_[set|assign|free|stringify]_id` and `lv_obj_id_compare`should be implemented externally.

Parametri

- **class_p** -- the class this obj belongs to. Note obj->class_p is the class currently being constructed.

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

void **lv_obj_free_id**(lv_obj_t *obj)

Free resources allocated by `lv_obj_assign_id` or `lv_obj_set_id`. This function is also called automatically when object is deleted.

Parametri

obj -- pointer to an object

int **lv_obj_id_compare**(const void *id1, const void *id2)

Compare two obj id, return 0 if they are equal.

Set `LV_USE_OBJ_ID_BUILTIN` to use the builtin method for compare. Otherwise, it must be implemented externally.

Parametri

- **id1** -- the first id
- **id2** -- the second id

Ritorna

0 if they are equal, non-zero otherwise.

const char ***lv_obj_stringify_id**(lv_obj_t *obj, char *buf, uint32_t len)

Format an object's id into a string.

Parametri

- **obj** -- pointer to an object
- **buf** -- buffer to write the string into
- **len** -- length of the buffer

void **lv_objid_builtin_destroy**(void)

Free resources used by builtin ID generator.

Variables

const lv_obj_class_t **lv_obj_class**

Make the base object's class publicly available.

14.10.5 lv_obj_class.h

Typedef

typedef void (***lv_obj_class_event_cb_t**)(lv_obj_class_t *class_p, lv_event_t *e)

Enums

enum **lv_obj_class_editable_t**

Values:

enumerator **LV_OBJ_CLASS_EDITABLE_INHERIT**

Check the base class. Must have 0 value to let zero initialized class inherit

enumerator **LV_OBJ_CLASS_EDITABLE_TRUE**

enumerator **LV_OBJ_CLASS_EDITABLE_FALSE**

enum **lv_obj_class_group_def_t**

Values:

enumerator **LV_OBJ_CLASS_GROUP_DEF_INHERIT**

Check the base class. Must have 0 value to let zero initialized class inherit

enumerator **LV_OBJ_CLASS_GROUP_DEF_TRUE**

enumerator **LV_OBJ_CLASS_GROUP_DEF_FALSE**

enum **lv_obj_class_theme_inheritable_t**

Values:

enumerator **LV_OBJ_CLASS_THEME_INHERITABLE_FALSE**

Do not inherit theme from base class.

enumerator **LV_OBJ_CLASS_THEME_INHERITABLE_TRUE**

Functions

lv_obj_t ***lv_obj_class_create_obj**(const lv_obj_class_t *class_p, lv_obj_t *parent)

Create an object form a class descriptor

Parametri

- **class_p** -- pointer to a class
- **parent** -- pointer to an object where the new object should be created

Ritorna

pointer to the created object

void **lv_obj_class_init_obj**(lv_obj_t *obj)

bool **lv_obj_is_editable**(lv_obj_t *obj)

bool **lv_obj_is_group_def**(lv_obj_t *obj)

14.10.6 lv_obj_class_private.h

Functions

void **lv_obj_destruct**(lv_obj_t *obj)

struct **_lv_obj_class_t**

#include <lv_obj_class_private.h> Describe the common methods of every object. Similar to a C++ class.

Public Members

const lv_obj_class_t ***base_class**

void (***constructor_cb**)(const lv_obj_class_t *class_p, lv_obj_t *obj)

class_p is the final class while obj->class_p is the class currently being [de]constructed.

void (***destructor_cb**)(const lv_obj_class_t *class_p, lv_obj_t *obj)

void (***event_cb**)(const lv_obj_class_t *class_p, lv_event_t *e)

class_p is the class in which event is being processed. Widget type specific event function

uint32_t **prop_index_start**

uint32_t **prop_index_end**

const lv_property_ops_t ***properties**

uint32_t **properties_count**

const lv_property_name_t ***property_names**

uint32_t **names_count**

void ***user_data**

const char ***name**

int32_t **width_def**

int32_t **height_def**

uint32_t **editable**

Value from *lv_obj_class_editable_t*

uint32_t **group_def**

Value from *lv_obj_class_group_def_t*

uint32_t **instance_size**

uint32_t **theme_inheritable**

Value from *lv_obj_class_theme_inheritable_t*

14.10.7 lv_obj_draw.h

Enums

enum **lv_layer_type_t**

Store the type of layer required to render a widget.

Values:

enumerator **LV_LAYER_TYPE_NONE**

No layer is needed.

enumerator **LV_LAYER_TYPE_SIMPLE**

Simple layer means that the layer can be rendered in chunks. For example with opa_layered = 140 it's possible to render only 10 lines from the layer. When it's ready go the the next 10 lines. It avoids large memory allocations for the layer buffer. The buffer size for a chunk can be set by `LV_DRAW_LAYER_SIMPLE_BUF_SIZE` in lv_conf.h.

enumerator **LV_LAYER_TYPE_TRANSFORM**

The widget is transformed and cannot be rendered in chunks. It's because - due to the transformations - pixel outside of a given area will also contribute to the final image. In this case there is no limitation on the buffer size. LVGL will allocate as large buffer as needed to render the transformed area.

Functions

void **lv_obj_init_draw_rect_dsc**(lv_obj_t *obj, *lv_part_t* part, *lv_draw_rect_dsc_t* *draw_dsc)

Initialize a rectangle draw descriptor from an object's styles in its current state

> **Nota**
>
> Only the relevant fields will be set. E.g. if `border width == 0` the other border properties won't be evaluated.

Parametri

- **obj** -- pointer to an object
- **part** -- part of the object, e.g. `LV_PART_MAIN`, `LV_PART_SCROLLBAR`, `LV_PART_KNOB`, etc
- **draw_dsc** -- the descriptor to initialize. If an `..._opa` field is set to `LV_OPA_TRANSP` the related properties won't be initialized. Should be initialized with `lv_draw_rect_dsc_init(draw_dsc)`.

void **lv_obj_init_draw_label_dsc**(lv_obj_t *obj, *lv_part_t* part, *lv_draw_label_dsc_t* *draw_dsc)

Initialize a label draw descriptor from an object's styles in its current state

Parametri

- **obj** -- pointer to an object
- **part** -- part of the object, e.g. `LV_PART_MAIN`, `LV_PART_SCROLLBAR`, `LV_PART_KNOB`, etc
- **draw_dsc** -- the descriptor to initialize. If the `opa` field is set to or the property is equal to `LV_OPA_TRANSP` the rest won't be initialized. Should be initialized with `lv_draw_label_dsc_init(draw_dsc)`.

void **lv_obj_init_draw_image_dsc**(lv_obj_t *obj, *lv_part_t* part, lv_draw_image_dsc_t *draw_dsc)

Initialize an image draw descriptor from an object's styles in its current state

Parametri

- **obj** -- pointer to an object
- **part** -- part of the object, e.g. `LV_PART_MAIN`, `LV_PART_SCROLLBAR`, `LV_PART_KNOB`, etc
- **draw_dsc** -- the descriptor to initialize. Should be initialized with `lv_draw_image_dsc_init(draw_dsc)`.

void **lv_obj_init_draw_line_dsc**(lv_obj_t *obj, *lv_part_t* part, *lv_draw_line_dsc_t* *draw_dsc)

Initialize a line draw descriptor from an object's styles in its current state

Parametri

- **obj** -- pointer to an object
- **part** -- part of the object, e.g. `LV_PART_MAIN`, `LV_PART_SCROLLBAR`, `LV_PART_KNOB`, etc
- **draw_dsc** -- the descriptor to initialize. Should be initialized with `lv_draw_line_dsc_init(draw_dsc)`.

void **lv_obj_init_draw_arc_dsc**(lv_obj_t *obj, *lv_part_t* part, *lv_draw_arc_dsc_t* *draw_dsc)

Initialize an arc draw descriptor from an object's styles in its current state

Parametri

- **obj** -- pointer to an object
- **part** -- part of the object, e.g. `LV_PART_MAIN`, `LV_PART_SCROLLBAR`, `LV_PART_KNOB`, etc

- **draw_dsc** -- the descriptor to initialize. Should be initialized with `lv_draw_arc_dsc_init(draw_dsc)`.

int32_t **lv_obj_calculate_ext_draw_size**(lv_obj_t *obj, *lv_part_t* part)

Get the required extra size (around the object's part) to draw shadow, outline, value etc.

Parametri

- **obj** -- pointer to an object
- **part** -- part of the object

Ritorna

the extra size required around the object

void **lv_obj_refresh_ext_draw_size**(lv_obj_t *obj)

Send a 'LV_EVENT_REFR_EXT_DRAW_SIZE' Call the ancestor's event handler to the object to refresh the value of the extended draw size. The result will be saved in `obj`.

Parametri

obj -- pointer to an object

14.10.8 lv_obj_draw_private.h

Functions

int32_t **lv_obj_get_ext_draw_size**(const lv_obj_t *obj)

Get the extended draw area of an object.

Parametri

obj -- pointer to an object

Ritorna

the size extended draw area around the real coordinates

lv_layer_type_t **lv_obj_get_layer_type**(const lv_obj_t *obj)

14.10.9 lv_obj_event.h

Enums

enum **lv_cover_res_t**

Cover check results.

Values:

enumerator **LV_COVER_RES_COVER**

enumerator **LV_COVER_RES_NOT_COVER**

enumerator **LV_COVER_RES_MASKED**

Functions

lv_result_t **lv_obj_send_event**(lv_obj_t *obj, *lv_event_code_t* event_code, void *param)

Send an event to the object

Parametri

- **obj** -- pointer to an object
- **event_code** -- the type of the event from `lv_event_t`
- **param** -- arbitrary data depending on the widget type and the event. (Usually `NULL`)

Ritorna

LV_RESULT_OK: `obj` was not deleted in the event; LV_RESULT_INVALID: `obj` was deleted in the event_code

lv_result_t **lv_obj_event_base**(const lv_obj_class_t *class_p, lv_event_t *e)

Used by the widgets internally to call the ancestor widget types's event handler

Parametri

- **class_p** -- pointer to the class of the widget (NOT the ancestor class)
- **e** -- pointer to the event descriptor

Ritorna

LV_RESULT_OK: the target object was not deleted in the event; LV_RESULT_INVALID: it was deleted in the event_code

lv_obj_t ***lv_event_get_current_target_obj**(lv_event_t *e)

Get the current target of the event. It's the object which event handler being called. If the event is not bubbled it's the same as "original" target.

Parametri

e -- pointer to the event descriptor

Ritorna

the target of the event_code

lv_obj_t ***lv_event_get_target_obj**(lv_event_t *e)

Get the object originally targeted by the event. It's the same even if the event is bubbled.

Parametri

e -- pointer to the event descriptor

Ritorna

pointer to the original target of the event_code

lv_event_dsc_t ***lv_obj_add_event_cb**(lv_obj_t *obj, *lv_event_cb_t* event_cb, *lv_event_code_t* filter, void *user_data)

Add an event handler function for an object. Used by the user to react on event which happens with the object. An object can have multiple event handler. They will be called in the same order as they were added.

Parametri

- **obj** -- pointer to an object
- **filter** -- an event code (e.g. `LV_EVENT_CLICKED`) on which the event should be called. `LV_EVENT_ALL` can be used to receive all the events.
- **event_cb** -- the new event function
- **user_data** -- custom data will be available in `event_cb`

Ritorna

handler to the event. It can be used in `lv_obj_remove_event_dsc`.

uint32_t **lv_obj_get_event_count**(lv_obj_t *obj)

lv_event_dsc_t ***lv_obj_get_event_dsc**(lv_obj_t *obj, uint32_t index)

bool **lv_obj_remove_event**(lv_obj_t *obj, uint32_t index)

bool **lv_obj_remove_event_dsc**(lv_obj_t *obj, lv_event_dsc_t *dsc)

uint32_t **lv_obj_remove_event_cb**(lv_obj_t *obj, *lv_event_cb_t* event_cb)

Remove an event_cb from an object

Parametri

- **obj** -- pointer to a obj
- **event_cb** -- the event_cb of the event to remove

Ritorna

the count of the event removed

uint32_t **lv_obj_remove_event_cb_with_user_data**(lv_obj_t *obj, *lv_event_cb_t* event_cb, void *user_data)

Remove an event_cb with user_data

Parametri

- **obj** -- pointer to a obj
- **event_cb** -- the event_cb of the event to remove
- **user_data** -- user_data

Ritorna
the count of the event removed

lv_indev_t ***lv_event_get_indev**(lv_event_t *e)

Get the input device passed as parameter to indev related events.

Parametri
e -- pointer to an event

Ritorna
the indev that triggered the event or NULL if called on a not indev related event

lv_layer_t ***lv_event_get_layer**(lv_event_t *e)

Get the draw context which should be the first parameter of the draw functions. Namely: `LV_EVENT_DRAW_MAIN/POST`, `LV_EVENT_DRAW_MAIN/POST_BEGIN`, `LV_EVENT_DRAW_MAIN/POST_END`

Parametri
e -- pointer to an event

Ritorna
pointer to a draw context or NULL if called on an unrelated event

const *lv_area_t* ***lv_event_get_old_size**(lv_event_t *e)

Get the old area of the object before its size was changed. Can be used in `LV_EVENT_SIZE_CHANGED`

Parametri
e -- pointer to an event

Ritorna
the old absolute area of the object or NULL if called on an unrelated event

uint32_t **lv_event_get_key**(lv_event_t *e)

Get the key passed as parameter to an event. Can be used in `LV_EVENT_KEY`

Parametri
e -- pointer to an event

Ritorna
the triggering key or NULL if called on an unrelated event

int32_t **lv_event_get_rotary_diff**(lv_event_t *e)

Get the signed rotary encoder diff. passed as parameter to an event. Can be used in `LV_EVENT_ROTARY`

Parametri
e -- pointer to an event

Ritorna
the triggering key or NULL if called on an unrelated event

lv_anim_t ***lv_event_get_scroll_anim**(lv_event_t *e)

Get the animation descriptor of a scrolling. Can be used in `LV_EVENT_SCROLL_BEGIN`

Parametri
e -- pointer to an event

Ritorna
the animation that will scroll the object. (can be modified as required)

void **lv_event_set_ext_draw_size**(lv_event_t *e, int32_t size)

Set the new extra draw size. Can be used in `LV_EVENT_REFR_EXT_DRAW_SIZE`

Parametri
- **e** -- pointer to an event
- **size** -- The new extra draw size

lv_point_t ***lv_event_get_self_size_info**(lv_event_t *e)

Get a pointer to an *`lv_point_t`* variable in which the self size should be saved (width in `point->x` and height `point->y`). Can be used in `LV_EVENT_GET_SELF_SIZE`

Parametri
e -- pointer to an event

Ritorna
pointer to *`lv_point_t`* or NULL if called on an unrelated event

lv_hit_test_info_t ***lv_event_get_hit_test_info**(lv_event_t *e)

Get a pointer to an `lv_hit_test_info_t` variable in which the hit test result should be saved. Can be used in `LV_EVENT_HIT_TEST`

Parametri

e -- pointer to an event

Ritorna

pointer to `lv_hit_test_info_t` or NULL if called on an unrelated event

const *lv_area_t* ***lv_event_get_cover_area**(lv_event_t *e)

Get a pointer to an area which should be examined whether the object fully covers it or not. Can be used in `LV_EVENT_HIT_TEST`

Parametri

e -- pointer to an event

Ritorna

an area with absolute coordinates to check

void **lv_event_set_cover_res**(lv_event_t *e, *lv_cover_res_t* res)

Set the result of cover checking. Can be used in `LV_EVENT_COVER_CHECK`

Parametri

- **e** -- pointer to an event
- **res** -- an element of ::lv_cover_check_info_t

lv_draw_task_t ***lv_event_get_draw_task**(lv_event_t *e)

Get the draw task which was just added. Can be used in `LV_EVENT_DRAW_TASK_ADDED event`

Parametri

e -- pointer to an event

Ritorna

the added draw task

14.10.10 lv_obj_event_private.h

struct **_lv_hit_test_info_t**

#include <lv_obj_event_private.h> Used as the event parameter of *LV_EVENT_HIT_TEST* to check if an `point` can click the object or not. `res` should be set like this:

- If already set to `false` another event wants that point non clickable. If you want to respect it leave it as `false` or set `true` to overwrite it.
- If already set `true` and `point` shouldn't be clickable set to `false`
- If already set to `true` you agree that `point` can click the object leave it as `true`

Public Members

const *lv_point_t* ***point**

A point relative to screen to check if it can click the object or not

bool **res**

true: `point` can click the object; false: it cannot

struct **_lv_cover_check_info_t**

#include <lv_obj_event_private.h> Used as the event parameter of *LV_EVENT_COVER_CHECK* to check if an area is covered by the object or not. In the event use `const lv_area_t * area = lv_event_get_cover_area(e)` to get the area to check and `lv_event_set_cover_res(e, res)` to set the result.

Public Members

lv_cover_res_t **res**

const *lv_area_t* ***area**

14.10.11 lv_obj_pos.h

Enums

enum **lv_obj_point_transform_flag_t**

Values:

enumerator **LV_OBJ_POINT_TRANSFORM_FLAG_NONE**

No flags

enumerator **LV_OBJ_POINT_TRANSFORM_FLAG_RECURSIVE**

Consider the transformation properties of the parents too

enumerator **LV_OBJ_POINT_TRANSFORM_FLAG_INVERSE**

Execute the inverse of the transformation (-angle and 1/zoom)

enumerator **LV_OBJ_POINT_TRANSFORM_FLAG_INVERSE_RECURSIVE**

Both inverse and recursive

Functions

void **lv_obj_set_pos**(lv_obj_t *obj, int32_t x, int32_t y)

Set the position of an object relative to the set alignment.

> **Nota**
>
> With default alignment it's the distance from the top left corner

> **Nota**
>
> E.g. LV_ALIGN_CENTER alignment it's the offset from the center of the parent

> **Nota**
>
> The position is interpreted on the content area of the parent

> **Nota**
>
> The values can be set in pixel or in percentage of parent size with `lv_pct(v)`

Parametri

- **obj** -- pointer to an object
- **x** -- new x coordinate

- **y** -- new y coordinate

void **lv_obj_set_x**(lv_obj_t *obj, int32_t x)

Set the x coordinate of an object

> **Nota**
>
> With default alignment it's the distance from the top left corner

> **Nota**
>
> E.g. LV_ALIGN_CENTER alignment it's the offset from the center of the parent

> **Nota**
>
> The position is interpreted on the content area of the parent

> **Nota**
>
> The values can be set in pixel or in percentage of parent size with `lv_pct(v)`

Parametri

- **obj** -- pointer to an object
- **x** -- new x coordinate

void **lv_obj_set_y**(lv_obj_t *obj, int32_t y)

Set the y coordinate of an object

> **Nota**
>
> With default alignment it's the distance from the top left corner

> **Nota**
>
> E.g. LV_ALIGN_CENTER alignment it's the offset from the center of the parent

> **Nota**
>
> The position is interpreted on the content area of the parent

> **Nota**
>
> The values can be set in pixel or in percentage of parent size with `lv_pct(v)`

Parametri

- **obj** -- pointer to an object
- **y** -- new y coordinate

void **lv_obj_set_size**(lv_obj_t *obj, int32_t w, int32_t h)

Set the size of an object.

> **Nota**
>
> possible values are: pixel simple set the size accordingly LV_SIZE_CONTENT set the size to involve all children in the given direction lv_pct(x) to set size in percentage of the parent's content area size (the size without paddings). x should be in [0..1000]% range

Parametri

- **obj** -- pointer to an object
- **w** -- the new width
- **h** -- the new height

bool **lv_obj_refr_size**(lv_obj_t *obj)

Recalculate the size of the object

Parametri

obj -- pointer to an object

Ritorna

true: the size has been changed

void **lv_obj_set_width**(lv_obj_t *obj, int32_t w)

Set the width of an object

> **Nota**
>
> possible values are: pixel simple set the size accordingly LV_SIZE_CONTENT set the size to involve all children in the given direction lv_pct(x) to set size in percentage of the parent's content area size (the size without paddings). x should be in [0..1000]% range

Parametri

- **obj** -- pointer to an object
- **w** -- the new width

void **lv_obj_set_height**(lv_obj_t *obj, int32_t h)

Set the height of an object

> **Nota**
>
> possible values are: pixel simple set the size accordingly LV_SIZE_CONTENT set the size to involve all children in the given direction lv_pct(x) to set size in percentage of the parent's content area size (the size without paddings). x should be in [0..1000]% range

Parametri

- **obj** -- pointer to an object
- **h** -- the new height

void **lv_obj_set_content_width**(lv_obj_t *obj, int32_t w)

Set the width reduced by the left and right padding and the border width.

Parametri

- **obj** -- pointer to an object
- **w** -- the width without paddings in pixels

void **lv_obj_set_content_height**(lv_obj_t *obj, int32_t h)

Set the height reduced by the top and bottom padding and the border width.

Parametri

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

- **h** -- the height without paddings in pixels

void **lv_obj_set_layout**(lv_obj_t *obj, uint32_t layout)

Set a layout for an object

Parametri

- **obj** -- pointer to an object
- **layout** -- pointer to a layout descriptor to set

bool **lv_obj_is_layout_positioned**(const lv_obj_t *obj)

Test whether the and object is positioned by a layout or not

Parametri

obj -- pointer to an object to test

Ritorna

true: positioned by a layout; false: not positioned by a layout

void **lv_obj_mark_layout_as_dirty**(lv_obj_t *obj)

Mark the object for layout update.

Parametri

obj -- pointer to an object whose children need to be updated

void **lv_obj_update_layout**(const lv_obj_t *obj)

Update the layout of an object.

Parametri

obj -- pointer to an object whose position and size needs to be updated

void **lv_obj_set_align**(lv_obj_t *obj, *lv_align_t* align)

Change the alignment of an object.

Parametri

- **obj** -- pointer to an object to align
- **align** -- type of alignment (see '*lv_align_t*' enum) `LV_ALIGN_OUT_...` can't be used.

void **lv_obj_align**(lv_obj_t *obj, *lv_align_t* align, int32_t x_ofs, int32_t y_ofs)

Change the alignment of an object and set new coordinates. Equivalent to: lv_obj_set_align(obj, align); lv_obj_set_pos(obj, x_ofs, y_ofs);

Parametri

- **obj** -- pointer to an object to align
- **align** -- type of alignment (see '*lv_align_t*' enum) `LV_ALIGN_OUT_...` can't be used.
- **x_ofs** -- x coordinate offset after alignment
- **y_ofs** -- y coordinate offset after alignment

void **lv_obj_align_to**(lv_obj_t *obj, const lv_obj_t *base, *lv_align_t* align, int32_t x_ofs, int32_t y_ofs)

Align an object to another object.

> **Nota**
>
> if the position or size of `base` changes `obj` needs to be aligned manually again

Parametri

- **obj** -- pointer to an object to align
- **base** -- pointer to another object (if NULL `obj`s parent is used). 'obj' will be aligned to it.
- **align** -- type of alignment (see '*lv_align_t*' enum)
- **x_ofs** -- x coordinate offset after alignment
- **y_ofs** -- y coordinate offset after alignment

void **lv_obj_center**(lv_obj_t *obj)

Align an object to the center on its parent.

> **Nota**
>
> if the parent size changes `obj` needs to be aligned manually again

Parametri
`obj` -- pointer to an object to align

void **`lv_obj_set_transform`**(lv_obj_t *obj, const lv_matrix_t *matrix)

Set the transform matrix of an object

> **Nota**
>
> `LV_DRAW_TRANSFORM_USE_MATRIX` needs to be enabled.

Parametri
- **`obj`** -- pointer to an object
- **`matrix`** -- pointer to a matrix to set

void **`lv_obj_reset_transform`**(lv_obj_t *obj)

Reset the transform matrix of an object to identity matrix

> **Nota**
>
> `LV_DRAW_TRANSFORM_USE_MATRIX` needs to be enabled.

Parametri
`obj` -- pointer to an object

void **`lv_obj_get_coords`**(const lv_obj_t *obj, *lv_area_t* *coords)

Copy the coordinates of an object to an area

Parametri
- **`obj`** -- pointer to an object
- **`coords`** -- pointer to an area to store the coordinates

int32_t **`lv_obj_get_x`**(const lv_obj_t *obj)

Get the x coordinate of object.

> **Nota**
>
> The position of the object is recalculated only on the next redraw. To force coordinate recalculation call `lv_obj_update_layout(obj)`.

> **Nota**
>
> Zero return value means the object is on the left padding of the parent, and not on the left edge.

> **Nota**
>
> Scrolling of the parent doesn't change the returned value.

> **Nota**
>
> The returned value is always the distance from the parent even if `obj` is positioned by a layout.

Parametri
`obj` -- pointer to an object
Ritorna
distance of `obj` from the left side of its parent plus the parent's left padding

int32_t **`lv_obj_get_x2`**(const lv_obj_t *obj)

Get the x2 coordinate of object.

> **Nota**
>
> The position of the object is recalculated only on the next redraw. To force coordinate recalculation call `lv_obj_update_layout(obj)`.

> **Nota**
>
> Zero return value means the object is on the right padding of the parent, and not on the right edge.

> **Nota**
>
> Scrolling of the parent doesn't change the returned value.

> **Nota**
>
> The returned value is always the distance from the parent even if `obj` is positioned by a layout.

Parametri
`obj` -- pointer to an object
Ritorna
distance of `obj` from the right side of its parent plus the parent's right padding

int32_t **`lv_obj_get_y`**(const lv_obj_t *obj)

Get the y coordinate of object.

> **Nota**
>
> The position of the object is recalculated only on the next redraw. To force coordinate recalculation call `lv_obj_update_layout(obj)`.

> **Nota**
>
> Zero return value means the object is on the top padding of the parent, and not on the top edge.

> **Nota**

Scrolling of the parent doesn't change the returned value.

Nota

The returned value is always the distance from the parent even if `obj` is positioned by a layout.

Parametri
`obj` -- pointer to an object
Ritorna
distance of `obj` from the top side of its parent plus the parent's top padding

int32_t **`lv_obj_get_y2`**(const lv_obj_t *obj)

Get the y2 coordinate of object.

Nota

The position of the object is recalculated only on the next redraw. To force coordinate recalculation call `lv_obj_update_layout(obj)`.

Nota

Zero return value means the object is on the bottom padding of the parent, and not on the bottom edge.

Nota

Scrolling of the parent doesn't change the returned value.

Nota

The returned value is always the distance from the parent even if `obj` is positioned by a layout.

Parametri
`obj` -- pointer to an object
Ritorna
distance of `obj` from the bottom side of its parent plus the parent's bottom padding

int32_t **`lv_obj_get_x_aligned`**(const lv_obj_t *obj)

Get the actually set x coordinate of object, i.e. the offset from the set alignment
Parametri
`obj` -- pointer to an object
Ritorna
the set x coordinate

int32_t **`lv_obj_get_y_aligned`**(const lv_obj_t *obj)

Get the actually set y coordinate of object, i.e. the offset from the set alignment
Parametri
`obj` -- pointer to an object
Ritorna
the set y coordinate

int32_t **lv_obj_get_width**(const lv_obj_t *obj)

Get the width of an object

> **Nota**
>
> The position of the object is recalculated only on the next redraw. To force coordinate recalculation call `lv_obj_update_layout(obj)`.

Parametri

obj -- pointer to an object

Ritorna

the width in pixels

int32_t **lv_obj_get_height**(const lv_obj_t *obj)

Get the height of an object

> **Nota**
>
> The position of the object is recalculated only on the next redraw. To force coordinate recalculation call `lv_obj_update_layout(obj)`.

Parametri

obj -- pointer to an object

Ritorna

the height in pixels

int32_t **lv_obj_get_content_width**(const lv_obj_t *obj)

Get the width reduced by the left and right padding and the border width.

> **Nota**
>
> The position of the object is recalculated only on the next redraw. To force coordinate recalculation call `lv_obj_update_layout(obj)`.

Parametri

obj -- pointer to an object

Ritorna

the width which still fits into its parent without causing overflow (making the parent scrollable)

int32_t **lv_obj_get_content_height**(const lv_obj_t *obj)

Get the height reduced by the top and bottom padding and the border width.

> **Nota**
>
> The position of the object is recalculated only on the next redraw. To force coordinate recalculation call `lv_obj_update_layout(obj)`.

Parametri

obj -- pointer to an object

Ritorna

the height which still fits into the parent without causing overflow (making the parent scrollable)

void **lv_obj_get_content_coords**(const lv_obj_t *obj, *lv_area_t* *area)

Get the area reduced by the paddings and the border width.

> **Nota**
>
> The position of the object is recalculated only on the next redraw. To force coordinate recalculation call `lv_obj_update_layout(obj)`.

Parametri

- **obj** -- pointer to an object
- **area** -- the area which still fits into the parent without causing overflow (making the parent scrollable)

int32_t **lv_obj_get_self_width**(const lv_obj_t *obj)

Get the width occupied by the "parts" of the widget. E.g. the width of all columns of a table.

> **Nota**
>
> This size independent from the real size of the widget. It just tells how large the internal ("virtual") content is.

Parametri

obj -- pointer to an objects

Ritorna

the width of the virtually drawn content

int32_t **lv_obj_get_self_height**(const lv_obj_t *obj)

Get the height occupied by the "parts" of the widget. E.g. the height of all rows of a table.

> **Nota**
>
> This size independent from the real size of the widget. It just tells how large the internal ("virtual") content is.

Parametri

obj -- pointer to an objects

Ritorna

the width of the virtually drawn content

bool **lv_obj_refresh_self_size**(lv_obj_t *obj)

Handle if the size of the internal ("virtual") content of an object has changed.

Parametri

obj -- pointer to an object

Ritorna

false: nothing happened; true: refresh happened

void **lv_obj_refr_pos**(lv_obj_t *obj)

void **lv_obj_move_to**(lv_obj_t *obj, int32_t x, int32_t y)

void **lv_obj_move_children_by**(lv_obj_t *obj, int32_t x_diff, int32_t y_diff, bool ignore_floating)

const lv_matrix_t ***lv_obj_get_transform**(const lv_obj_t *obj)

Get the transform matrix of an object

Parametri

obj -- pointer to an object

Ritorna

pointer to the transform matrix or NULL if not set

void **lv_obj_transform_point**(const lv_obj_t *obj, *lv_point_t* *p, *lv_obj_point_transform_flag_t* flags)

Transform a point using the angle and zoom style properties of an object

Parametri

- **obj** -- pointer to an object whose style properties should be used
- **p** -- a point to transform, the result will be written back here too
- **flags** -- OR-ed valued of :cpp:enum:`lv_obj_point_transform_flag_t`

void **lv_obj_transform_point_array**(const lv_obj_t *obj, *lv_point_t* points[], size_t count, *lv_obj_point_transform_flag_t* flags)

Transform an array of points using the angle and zoom style properties of an object

Parametri

- **obj** -- pointer to an object whose style properties should be used
- **points** -- the array of points to transform, the result will be written back here too
- **count** -- number of points in the array
- **flags** -- OR-ed valued of :cpp:enum:`lv_obj_point_transform_flag_t`

void **lv_obj_get_transformed_area**(const lv_obj_t *obj, *lv_area_t* *area, *lv_obj_point_transform_flag_t* flags)

Transform an area using the angle and zoom style properties of an object

Parametri

- **obj** -- pointer to an object whose style properties should be used
- **area** -- an area to transform, the result will be written back here too
- **flags** -- OR-ed valued of :cpp:enum:`lv_obj_point_transform_flag_t`

void **lv_obj_invalidate_area**(const lv_obj_t *obj, const *lv_area_t* *area)

Mark an area of an object as invalid. The area will be truncated to the object's area and marked for redraw.

Parametri

- **obj** -- pointer to an object
- **area** -- the area to redraw

void **lv_obj_invalidate**(const lv_obj_t *obj)

Mark the object as invalid to redrawn its area

Parametri

obj -- pointer to an object

bool **lv_obj_area_is_visible**(const lv_obj_t *obj, *lv_area_t* *area)

Tell whether an area of an object is visible (even partially) now or not

Parametri

- **obj** -- pointer to an object
- **area** -- the are to check. The visible part of the area will be written back here.

Ritorna

true visible; false not visible (hidden, out of parent, on other screen, etc)

bool **lv_obj_is_visible**(const lv_obj_t *obj)

Tell whether an object is visible (even partially) now or not

Parametri

obj -- pointer to an object

Ritorna

true: visible; false not visible (hidden, out of parent, on other screen, etc)

void **lv_obj_set_ext_click_area**(lv_obj_t *obj, int32_t size)

Set the size of an extended clickable area

Parametri

- **obj** -- pointer to an object
- **size** -- extended clickable area in all 4 directions [px]

void **lv_obj_get_click_area**(const lv_obj_t *obj, *lv_area_t* *area)

Get the an area where to object can be clicked. It's the object's normal area plus the extended click area.

Parametri

- **obj** -- pointer to an object
- **area** -- store the result area here

bool **lv_obj_hit_test**(lv_obj_t *obj, const *lv_point_t* *point)

Hit-test an object given a particular point in screen space.

Parametri
- **obj** -- object to hit-test
- **point** -- screen-space point (absolute coordinate)

Ritorna
true: if the object is considered under the point

int32_t **lv_clamp_width**(int32_t width, int32_t min_width, int32_t max_width, int32_t ref_width)

Clamp a width between min and max width. If the min/max width is in percentage value use the ref_width

Parametri
- **width** -- width to clamp
- **min_width** -- the minimal width
- **max_width** -- the maximal width
- **ref_width** -- the reference width used when min/max width is in percentage

Ritorna
the clamped width

int32_t **lv_clamp_height**(int32_t height, int32_t min_height, int32_t max_height, int32_t ref_height)

Clamp a height between min and max height. If the min/max height is in percentage value use the ref_height

Parametri
- **height** -- height to clamp
- **min_height** -- the minimal height
- **max_height** -- the maximal height
- **ref_height** -- the reference height used when min/max height is in percentage

Ritorna
the clamped height

14.10.12 lv_obj_private.h

struct **_lv_obj_spec_attr_t**

#include <lv_obj_private.h> Special, rarely used attributes. They are allocated automatically if any elements is set.

Public Members

lv_obj_t ****children**

Store the pointer of the children in an array.

lv_group_t ***group_p**

lv_event_list_t **event_list**

const char ***name**

Pointer to the name

lv_point_t **scroll**

The current X/Y scroll offset

int32_t **ext_click_pad**

Extra click padding in all direction

int32_t **ext_draw_size**

EXTend the size in every direction for drawing.

uint16_t **child_cnt**

Number of children

uint16_t **scrollbar_mode**

How to display scrollbars, see *lv_scrollbar_mode_t*

uint16_t **scroll_snap_x**

Where to align the snappable children horizontally, see *lv_scroll_snap_t*

uint16_t **scroll_snap_y**

Where to align the snappable children vertically

uint16_t **scroll_dir**

The allowed scroll direction(s), see `lv_dir_t`

uint16_t **layer_type**

Cache the layer type here. Element of lv_intermediate_layer_type_t

uint16_t **name_static**

1: `name` was not dynamically allocated

struct **_lv_obj_t**

Public Members

const lv_obj_class_t ***class_p**

lv_obj_t ***parent**

lv_obj_spec_attr_t ***spec_attr**

lv_obj_style_t ***styles**

void ***user_data**

void ***id**

lv_area_t **coords**

lv_obj_flag_t **flags**

uint16_t **state**

uint16_t **layout_inv**

uint16_t **readjust_scroll_after_layout**

uint16_t **scr_layout_inv**

uint16_t **skip_trans**

uint16_t **style_cnt**

uint16_t **h_layout**

uint16_t **w_layout**

uint16_t **is_deleting**

14.10.13 lv_obj_scroll.h

Enums

enum **lv_scrollbar_mode_t**

Scrollbar modes: shows when should the scrollbars be visible

Values:

enumerator **LV_SCROLLBAR_MODE_OFF**

Never show scrollbars

enumerator **LV_SCROLLBAR_MODE_ON**

Always show scrollbars

enumerator **LV_SCROLLBAR_MODE_ACTIVE**

Show scroll bars when Widget is being scrolled

enumerator **LV_SCROLLBAR_MODE_AUTO**

Show scroll bars when the content is large enough to be scrolled

enum **lv_scroll_snap_t**

Scroll span align options. Tells where to align the snappable children when scroll stops.

Values:

enumerator **LV_SCROLL_SNAP_NONE**

Do not align, leave where it is

enumerator **LV_SCROLL_SNAP_START**

Align to the left/top

enumerator **LV_SCROLL_SNAP_END**

Align to the right/bottom

enumerator **LV_SCROLL_SNAP_CENTER**

Align to the center

Functions

void **lv_obj_set_scrollbar_mode**(lv_obj_t *obj, *lv_scrollbar_mode_t* mode)

Set how the scrollbars should behave.

Parametri

- **obj** -- pointer to Widget
- **mode** -- LV_SCROLL_MODE_ON/OFF/AUTO/ACTIVE

void **lv_obj_set_scroll_dir**(lv_obj_t *obj, *lv_dir_t* dir)

Set direction Widget can be scrolled

Parametri

- **obj** -- pointer to Widget
- **dir** -- one or more bit-wise OR-ed values of `lv_dir_t` enumeration

void **lv_obj_set_scroll_snap_x**(lv_obj_t *obj, *lv_scroll_snap_t* align)

Set where to snap the children when scrolling ends horizontally

Parametri

- **obj** -- pointer to Widget
- **align** -- value from `lv_scroll_snap_t` enumeration

void **lv_obj_set_scroll_snap_y**(lv_obj_t *obj, *lv_scroll_snap_t* align)

Set where to snap the children when scrolling ends vertically

Parametri

- **obj** -- pointer to Widget
- **align** -- value from `lv_scroll_snap_t` enumeration

lv_scrollbar_mode_t **lv_obj_get_scrollbar_mode**(const lv_obj_t *obj)

Get the current scroll mode (when to hide the scrollbars)

Parametri

obj -- pointer to Widget

Ritorna

the current scroll mode from `lv_scrollbar_mode_t`

lv_dir_t **lv_obj_get_scroll_dir**(const lv_obj_t *obj)

Get directions Widget can be scrolled (set with `lv_obj_set_scroll_dir()`)

Parametri

obj -- pointer to Widget

Ritorna

current scroll direction bit(s)

lv_scroll_snap_t **lv_obj_get_scroll_snap_x**(const lv_obj_t *obj)

Get where to snap child Widgets when horizontal scrolling ends.

Parametri

obj -- pointer to Widget

Ritorna

current snap value from `lv_scroll_snap_t`

lv_scroll_snap_t **lv_obj_get_scroll_snap_y**(const lv_obj_t *obj)

Get where to snap child Widgets when vertical scrolling ends.

Parametri

obj -- pointer to Widget

Ritorna

current snap value from `lv_scroll_snap_t`

int32_t **lv_obj_get_scroll_x**(const lv_obj_t *obj)

Get current X scroll position. Identical to `lv_obj_get_scroll_left()`.

Parametri

obj -- pointer to scrollable container Widget

Ritorna

current scroll position from left edge

- If Widget is not scrolled return 0.
- If scrolled return > 0.
- If scrolled inside (elastic scroll) return < 0.

int32_t **lv_obj_get_scroll_y**(const lv_obj_t *obj)

Get current Y scroll position. Identical to *lv_obj_get_scroll_top()*.

Parametri

obj -- pointer to scrollable container Widget

Ritorna

current scroll position from top edge

- If Widget is not scrolled return 0.
- If scrolled return > 0.
- If scrolled inside (elastic scroll) return < 0.

int32_t **lv_obj_get_scroll_top**(const lv_obj_t *obj)

Number of pixels a scrollable container Widget can be scrolled down before its top edge appears. When LV_OBJ_FLAG_SCROLL_ELASTIC flag is set in Widget, this value can go negative while Widget is being dragged below its normal top-edge boundary.

Parametri

obj -- pointer to scrollable container Widget

Ritorna

pixels Widget can be scrolled down before its top edge appears

int32_t **lv_obj_get_scroll_bottom**(const lv_obj_t *obj)

Number of pixels a scrollable container Widget can be scrolled up before its bottom edge appears. When LV_OBJ_FLAG_SCROLL_ELASTIC flag is set in Widget, this value can go negative while Widget is being dragged above its normal bottom-edge boundary.

Parametri

obj -- pointer to scrollable container Widget

Ritorna

pixels Widget can be scrolled up before its bottom edge appears

int32_t **lv_obj_get_scroll_left**(const lv_obj_t *obj)

Number of pixels a scrollable container Widget can be scrolled right before its left edge appears. When LV_OBJ_FLAG_SCROLL_ELASTIC flag is set in Widget, this value can go negative while Widget is being dragged farther right than its normal left-edge boundary.

Parametri

obj -- pointer to scrollable container Widget

Ritorna

pixels Widget can be scrolled right before its left edge appears

int32_t **lv_obj_get_scroll_right**(const lv_obj_t *obj)

Number of pixels a scrollable container Widget can be scrolled left before its right edge appears. When LV_OBJ_FLAG_SCROLL_ELASTIC flag is set in Widget, this value can go negative while Widget is being dragged farther left than its normal right-edge boundary.

Parametri

obj -- pointer to scrollable container Widget

Ritorna

pixels Widget can be scrolled left before its right edge appears

void **lv_obj_get_scroll_end**(lv_obj_t *obj, *lv_point_t* *end)

Get the X and Y coordinates where the scrolling will end for Widget if a scrolling animation is in progress. If no scrolling animation, give the current `x` or `y` scroll position.

Parametri

- **obj** -- pointer to scrollable Widget
- **end** -- pointer to *lv_point_t* object in which to store result

void **lv_obj_scroll_by**(lv_obj_t *obj, int32_t dx, int32_t dy, *lv_anim_enable_t* anim_en)

Scroll by given amount of pixels.

> **Nota**
>
> > 0 value means scroll right/bottom (show the more content on the right/bottom)

> **Nota**
>
> e.g. dy = -20 means scroll down 20 px

Parametri

- **obj** -- pointer to scrollable Widget to scroll
- **dx** -- pixels to scroll horizontally
- **dy** -- pixels to scroll vertically
- **anim_en** -- LV_ANIM_ON: scroll with animation; LV_ANIM_OFF: scroll immediately

void **lv_obj_scroll_by_bounded**(lv_obj_t *obj, int32_t dx, int32_t dy, *lv_anim_enable_t* anim_en)

Scroll by given amount of pixels. `dx` and `dy` will be limited internally to allow scrolling only on the content area.

> **Nota**
>
> e.g. dy = -20 means scroll down 20 px

Parametri

- **obj** -- pointer to scrollable Widget to scroll
- **dx** -- pixels to scroll horizontally
- **dy** -- pixels to scroll vertically
- **anim_en** -- LV_ANIM_ON: scroll with animation; LV_ANIM_OFF: scroll immediately

void **lv_obj_scroll_to**(lv_obj_t *obj, int32_t x, int32_t y, *lv_anim_enable_t* anim_en)

Scroll to given coordinate on Widget. `x` and `y` will be limited internally to allow scrolling only on the content area.

Parametri

- **obj** -- pointer to scrollable Widget to scroll
- **x** -- pixels to scroll horizontally
- **y** -- pixels to scroll vertically
- **anim_en** -- LV_ANIM_ON: scroll with animation; LV_ANIM_OFF: scroll immediately

void **lv_obj_scroll_to_x**(lv_obj_t *obj, int32_t x, *lv_anim_enable_t* anim_en)

Scroll to X coordinate on Widget. `x` will be limited internally to allow scrolling only on the content area.

Parametri

- **obj** -- pointer to scrollable Widget to scroll
- **x** -- pixels to scroll horizontally
- **anim_en** -- LV_ANIM_ON: scroll with animation; LV_ANIM_OFF: scroll immediately

void **lv_obj_scroll_to_y**(lv_obj_t *obj, int32_t y, *lv_anim_enable_t* anim_en)

Scroll to Y coordinate on Widget. `y` will be limited internally to allow scrolling only on the content area.

Parametri

- **obj** -- pointer to scrollable Widget to scroll
- **y** -- pixels to scroll vertically
- **anim_en** -- LV_ANIM_ON: scroll with animation; LV_ANIM_OFF: scroll immediately

void **lv_obj_scroll_to_view**(lv_obj_t *obj, *lv_anim_enable_t* anim_en)

Scroll `obj`'s parent Widget until `obj` becomes visible.

Parametri

- **obj** -- pointer to Widget to scroll into view
- **anim_en** -- LV_ANIM_ON: scroll with animation; LV_ANIM_OFF: scroll immediately

void **lv_obj_scroll_to_view_recursive**(lv_obj_t *obj, *lv_anim_enable_t* anim_en)

Scroll `obj`'s parent Widgets recursively until `obj` becomes visible. Widget will be scrolled into view even it has nested scrollable parents.

Parametri

- **obj** -- pointer to Widget to scroll into view
- **anim_en** -- LV_ANIM_ON: scroll with animation; LV_ANIM_OFF: scroll immediately

bool **lv_obj_is_scrolling**(const lv_obj_t *obj)

Tell whether Widget is being scrolled or not at this moment

Parametri

obj -- pointer to Widget

Ritorna

true: `obj` is being scrolled

void **lv_obj_stop_scroll_anim**(const lv_obj_t *obj)

Stop scrolling the current object

Parametri

obj -- The object being scrolled

void **lv_obj_update_snap**(lv_obj_t *obj, *lv_anim_enable_t* anim_en)

Check children of `obj` and scroll `obj` to fulfill scroll_snap settings.

Parametri

- **obj** -- Widget whose children need to be checked and snapped
- **anim_en** -- LV_ANIM_ON/OFF

void **lv_obj_get_scrollbar_area**(lv_obj_t *obj, *lv_area_t* *hor, *lv_area_t* *ver)

Get the area of the scrollbars

Parametri

- **obj** -- pointer to Widget
- **hor** -- pointer to store the area of the horizontal scrollbar
- **ver** -- pointer to store the area of the vertical scrollbar

void **lv_obj_scrollbar_invalidate**(lv_obj_t *obj)

Invalidate the area of the scrollbars

Parametri

obj -- pointer to Widget

void **lv_obj_readjust_scroll**(lv_obj_t *obj, *lv_anim_enable_t* anim_en)

Checks if the content is scrolled "in" and adjusts it to a normal position.

Parametri

- **obj** -- pointer to Widget
- **anim_en** -- LV_ANIM_ON/OFF

14.10.14 lv_obj_scroll_private.h

Functions

lv_result_t **lv_obj_scroll_by_raw**(lv_obj_t *obj, int32_t x, int32_t y)

Low level function to scroll by given x and y coordinates. `LV_EVENT_SCROLL` is sent.

Parametri

- **obj** -- pointer to an object to scroll
- **x** -- pixels to scroll horizontally

- **y** -- pixels to scroll vertically

Ritorna

`LV_RESULT_INVALID`: to object was deleted in `LV_EVENT_SCROLL`; `LV_RESULT_OK`: if the object is still valid

14.10.15 lv_obj_style.h

Typedef

typedef uint32_t **`lv_style_selector_t`**

A joint type for *`lv_part_t`* and *`lv_state_t`*. Example values

- `0`: means `LV_PART_MAIN | LV_STATE_DEFAULT`
- `LV_STATE_PRSSED`
- `LV_PART_KNOB`
- `LV_PART_KNOB | LV_STATE_PRESSED | LV_STATE_CHECKED`

Enums

enum **`lv_state_t`**

Possible states of a widget. OR-ed values are possible

Values:

enumerator **`LV_STATE_DEFAULT`**

enumerator **`LV_STATE_CHECKED`**

enumerator **`LV_STATE_FOCUSED`**

enumerator **`LV_STATE_FOCUS_KEY`**

enumerator **`LV_STATE_EDITED`**

enumerator **`LV_STATE_HOVERED`**

enumerator **`LV_STATE_PRESSED`**

enumerator **`LV_STATE_SCROLLED`**

enumerator **`LV_STATE_DISABLED`**

enumerator **`LV_STATE_USER_1`**

enumerator **`LV_STATE_USER_2`**

enumerator **`LV_STATE_USER_3`**

enumerator **`LV_STATE_USER_4`**

enumerator **LV_STATE_ANY**

Special value can be used in some functions to target all states

enum **lv_part_t**

The possible parts of widgets. The parts can be considered as the internal building block of the widgets. E.g. slider = background + indicator + knob Not all parts are used by every widget

Values:

enumerator **LV_PART_MAIN**

A background like rectangle

enumerator **LV_PART_SCROLLBAR**

The scrollbar(s)

enumerator **LV_PART_INDICATOR**

Indicator, e.g. for slider, bar, switch, or the tick box of the checkbox

enumerator **LV_PART_KNOB**

Like handle to grab to adjust the value

enumerator **LV_PART_SELECTED**

Indicate the currently selected option or section

enumerator **LV_PART_ITEMS**

Used if the widget has multiple similar elements (e.g. table cells)

enumerator **LV_PART_CURSOR**

Mark a specific place e.g. for text area's cursor or on a chart

enumerator **LV_PART_CUSTOM_FIRST**

Extension point for custom widgets

enumerator **LV_PART_ANY**

Special value can be used in some functions to target all parts

enum **lv_style_state_cmp_t**

Values:

enumerator **LV_STYLE_STATE_CMP_SAME**

The style properties in the 2 states are identical

enumerator **LV_STYLE_STATE_CMP_DIFF_REDRAW**

The differences can be shown with a simple redraw

enumerator **LV_STYLE_STATE_CMP_DIFF_DRAW_PAD**

The differences can be shown with a simple redraw

enumerator **LV_STYLE_STATE_CMP_DIFF_LAYOUT**

The differences can be shown with a simple redraw

Functions

void **lv_obj_add_style**(lv_obj_t *obj, const *lv_style_t* *style, *lv_style_selector_t* selector)

Add a style to an object.

Examples:

```
lv_obj_add_style(btn, &style_btn, 0); //Default button style

lv_obj_add_style(btn, &btn_red, LV_STATE_PRESSED); //Overwrite only␣
↪some colors to red when pressed
```

Parametri

- **obj** -- pointer to an object
- **style** -- pointer to a style to add
- **selector** -- OR-ed value of parts and state to which the style should be added

bool **lv_obj_replace_style**(lv_obj_t *obj, const *lv_style_t* *old_style, const *lv_style_t* *new_style, *lv_style_selector_t* selector)

Replaces a style of an object, preserving the order of the style stack (local styles and transitions are ignored).

Examples:

```
lv_obj_replace_style(obj, &yellow_style, &blue_style, LV_PART_ANY | LV_
↪STATE_ANY); //Replace a specific style

lv_obj_replace_style(obj, &yellow_style, &blue_style, LV_PART_MAIN |␣
↪LV_STATE_PRESSED); //Replace a specific style assigned to the main␣
↪part when it is pressed
```

Parametri

- **obj** -- pointer to an object
- **old_style** -- pointer to a style to replace.
- **new_style** -- pointer to a style to replace the old style with.
- **selector** -- OR-ed values of states and a part to replace only styles with matching selectors. LV_STATE_ANY and LV_PART_ANY can be used

void **lv_obj_remove_style**(lv_obj_t *obj, const *lv_style_t* *style, *lv_style_selector_t* selector)

Remove a style from an object.

Examples:

```
lv_obj_remove_style(obj, &style, LV_PART_ANY | LV_STATE_ANY); //Remove␣
↪a specific style

lv_obj_remove_style(obj, NULL, LV_PART_MAIN | LV_STATE_ANY); //Remove␣
↪all styles from the main part

lv_obj_remove_style(obj, NULL, LV_PART_ANY | LV_STATE_ANY); //Remove␣
↪all styles
```

Parametri

- **obj** -- pointer to an object
- **style** -- pointer to a style to remove. Can be NULL to check only the selector
- **selector** -- OR-ed values of states and a part to remove only styles with matching selectors. LV_STATE_ANY and LV_PART_ANY can be used

void **lv_obj_remove_style_all**(lv_obj_t *obj)

Remove all styles from an object

Parametri

obj -- pointer to an object

void **lv_obj_report_style_change**(*lv_style_t* *style)

Notify all object if a style is modified

Parametri

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

void **lv_obj_refresh_style**(lv_obj_t *obj, *lv_part_t* part, lv_style_prop_t prop)

Notify an object and its children about its style is modified.

Parametri

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

void **lv_obj_style_set_disabled**(lv_obj_t *obj, const *lv_style_t* *style, *lv_style_selector_t* selector, bool dis)

Temporary disable a style for a selector. It will look like is the style wasn't added

Parametri

- **obj** -- pointer to an object
- **style** -- pointer to a style
- **selector** -- the selector of a style (e.g. LV_STATE_PRESSED | LV_PART_KNOB)
- **dis** -- true: disable the style, false: enable the style

bool **lv_obj_style_get_disabled**(lv_obj_t *obj, const *lv_style_t* *style, *lv_style_selector_t* selector)

Get if a given style is disabled on an object.

Parametri

- **obj** -- pointer to an object
- **style** -- pointer to a style
- **selector** -- the selector of a style (e.g. LV_STATE_PRESSED | LV_PART_KNOB)

Ritorna

true: disable the style, false: enable the style

void **lv_obj_enable_style_refresh**(bool en)

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

Parametri

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

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

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

Parametri

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

Ritorna

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

bool **lv_obj_has_style_prop**(const lv_obj_t *obj, *lv_style_selector_t* selector, lv_style_prop_t prop)

Check if an object has a specified style property for a given style selector.

Parametri

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

- **selector** -- the style selector to be checked, defining the scope of the style to be examined.
- **prop** -- the property to be checked.

Ritorna

true if the object has the specified selector and property, false otherwise.

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

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

Parametri

- **obj** -- pointer to an object
- **prop** -- the property
- **value** -- value of the property. The correct element should be set according to the type of the property
- **selector** -- OR-ed value of parts and state for which the style should be set

lv_style_res_t **lv_obj_get_local_style_prop**(lv_obj_t *obj, lv_style_prop_t prop, *lv_style_value_t* *value, *lv_style_selector_t* selector)

bool **lv_obj_remove_local_style_prop**(lv_obj_t *obj, lv_style_prop_t prop, *lv_style_selector_t* selector)

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

Parametri

- **obj** -- pointer to an object
- **prop** -- a style property to remove.
- **selector** -- OR-ed value of parts and state for which the style should be removed

Ritorna

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

lv_style_value_t **lv_obj_style_apply_color_filter**(const lv_obj_t *obj, *lv_part_t* part, *lv_style_value_t* v)

Used internally for color filtering

void **lv_obj_fade_in**(lv_obj_t *obj, uint32_t time, uint32_t delay)

Fade in an an object and all its children.

Parametri

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

void **lv_obj_fade_out**(lv_obj_t *obj, uint32_t time, uint32_t delay)

Fade out an an object and all its children.

Parametri

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

static inline *lv_state_t* **lv_obj_style_get_selector_state**(*lv_style_selector_t* selector)

static inline *lv_part_t* **lv_obj_style_get_selector_part**(*lv_style_selector_t* selector)

static inline void **lv_obj_set_style_pad_all**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

static inline void **lv_obj_set_style_pad_hor**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

static inline void **lv_obj_set_style_pad_ver**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

static inline void **lv_obj_set_style_margin_all**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

static inline void **lv_obj_set_style_margin_hor**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

static inline void **lv_obj_set_style_margin_ver**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

static inline void **lv_obj_set_style_pad_gap**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

static inline void **lv_obj_set_style_size**(lv_obj_t *obj, int32_t width, int32_t height, *lv_style_selector_t* selector)

static inline void **lv_obj_set_style_transform_scale**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

static inline int32_t **lv_obj_get_style_space_left**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_space_right**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_space_top**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_space_bottom**(const lv_obj_t *obj, *lv_part_t* part)

lv_text_align_t **lv_obj_calculate_style_text_align**(const lv_obj_t *obj, *lv_part_t* part, const char *txt)

static inline int32_t **lv_obj_get_style_transform_scale_x_safe**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_transform_scale_y_safe**(const lv_obj_t *obj, *lv_part_t* part)

lv_opa_t **lv_obj_get_style_opa_recursive**(const lv_obj_t *obj, *lv_part_t* part)

Get the `opa` style property from all parents and multiply and `>> 8` them.

Parametri

- **obj** -- the object whose opacity should be get
- **part** -- the part whose opacity should be get. Non-MAIN parts will consider the `opa` of the MAIN part too

Ritorna

the final opacity considering the parents' opacity too

lv_color32_t **lv_obj_style_apply_recolor**(const lv_obj_t *obj, *lv_part_t* part, *lv_color32_t* color)

Apply recolor effect to the input color based on the object's style properties.

Parametri

- **obj** -- the target object containing recolor style properties
- **part** -- the part to retrieve recolor styles.
- **color** -- the original color to be modified

Ritorna

the blended color after applying recolor and opacity

lv_color32_t **lv_obj_get_style_recolor_recursive**(const lv_obj_t *obj, *lv_part_t* part)

Get the `recolor` style property from all parents and blend them recursively.

Parametri

- **obj** -- the object whose recolor value should be retrieved
- **part** -- the target part to check. Non-MAIN parts will also consider the `recolor` value from the MAIN part during calculation

Ritorna

the final blended recolor value combining all parent's recolor values

14.10.16 lv_obj_style_gen.h

Functions

static inline int32_t **lv_obj_get_style_width**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_min_width**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_max_width**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_height**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_min_height**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_max_height**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_length**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_x**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_y**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_align_t* **lv_obj_get_style_align**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_transform_width**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_transform_height**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_translate_x**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_translate_y**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_translate_radial**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_transform_scale_x**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_transform_scale_y**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_transform_rotation**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_transform_pivot_x**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_transform_pivot_y**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_transform_skew_x**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_transform_skew_y**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_pad_top**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_pad_bottom**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_pad_left**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_pad_right**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_pad_row**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_pad_column**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_pad_radial**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_margin_top**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_margin_bottom**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_margin_left**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_margin_right**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_bg_color**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_bg_color_filtered**(const lv_obj_t *obj, *lv_part_t* part)

static inline lv_opa_t **lv_obj_get_style_bg_opa**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_bg_grad_color**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_bg_grad_color_filtered**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_grad_dir_t* **lv_obj_get_style_bg_grad_dir**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_bg_main_stop**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_bg_grad_stop**(const lv_obj_t *obj, *lv_part_t* part)

static inline lv_opa_t **lv_obj_get_style_bg_main_opa**(const lv_obj_t *obj, *lv_part_t* part)

static inline lv_opa_t **lv_obj_get_style_bg_grad_opa**(const lv_obj_t *obj, *lv_part_t* part)

static inline const *lv_grad_dsc_t* ***lv_obj_get_style_bg_grad**(const lv_obj_t *obj, *lv_part_t* part)

static inline const void ***lv_obj_get_style_bg_image_src**(const lv_obj_t *obj, *lv_part_t* part)

static inline lv_opa_t **lv_obj_get_style_bg_image_opa**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_bg_image_recolor**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_bg_image_recolor_filtered**(const lv_obj_t *obj, *lv_part_t* part)

static inline lv_opa_t **lv_obj_get_style_bg_image_recolor_opa**(const lv_obj_t *obj, *lv_part_t* part)

static inline bool **lv_obj_get_style_bg_image_tiled**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_border_color**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_border_color_filtered**(const lv_obj_t *obj, *lv_part_t* part)

static inline lv_opa_t **lv_obj_get_style_border_opa**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_border_width**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_border_side_t* **lv_obj_get_style_border_side**(const lv_obj_t *obj, *lv_part_t* part)

static inline bool **lv_obj_get_style_border_post**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_outline_width**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_outline_color**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_outline_color_filtered**(const lv_obj_t *obj, *lv_part_t* part)

static inline lv_opa_t **lv_obj_get_style_outline_opa**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_outline_pad**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_shadow_width**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_shadow_offset_x**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_shadow_offset_y**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_shadow_spread**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_shadow_color**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_shadow_color_filtered**(const lv_obj_t *obj, *lv_part_t* part)

static inline lv_opa_t **lv_obj_get_style_shadow_opa**(const lv_obj_t *obj, *lv_part_t* part)

static inline lv_opa_t **lv_obj_get_style_image_opa**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_image_recolor**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_image_recolor_filtered**(const lv_obj_t *obj, *lv_part_t* part)

static inline lv_opa_t **lv_obj_get_style_image_recolor_opa**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_line_width**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_line_dash_width**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_line_dash_gap**(const lv_obj_t *obj, *lv_part_t* part)

static inline bool **lv_obj_get_style_line_rounded**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_line_color**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_line_color_filtered**(const lv_obj_t *obj, *lv_part_t* part)

static inline lv_opa_t **lv_obj_get_style_line_opa**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_arc_width**(const lv_obj_t *obj, *lv_part_t* part)

static inline bool **lv_obj_get_style_arc_rounded**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_arc_color**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_arc_color_filtered**(const lv_obj_t *obj, *lv_part_t* part)

static inline lv_opa_t **lv_obj_get_style_arc_opa**(const lv_obj_t *obj, *lv_part_t* part)

static inline const void ***lv_obj_get_style_arc_image_src**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_text_color**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_text_color_filtered**(const lv_obj_t *obj, *lv_part_t* part)

static inline lv_opa_t **lv_obj_get_style_text_opa**(const lv_obj_t *obj, *lv_part_t* part)

static inline const lv_font_t ***lv_obj_get_style_text_font**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_text_letter_space**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_text_line_space**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_text_decor_t* **lv_obj_get_style_text_decor**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_text_align_t* **lv_obj_get_style_text_align**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_text_outline_stroke_color**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_text_outline_stroke_color_filtered**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_text_outline_stroke_width**(const lv_obj_t *obj, *lv_part_t* part)

static inline lv_opa_t **lv_obj_get_style_text_outline_stroke_opa**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_radius**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_radial_offset**(const lv_obj_t *obj, *lv_part_t* part)

static inline bool **lv_obj_get_style_clip_corner**(const lv_obj_t *obj, *lv_part_t* part)

static inline lv_opa_t **lv_obj_get_style_opa**(const lv_obj_t *obj, *lv_part_t* part)

static inline lv_opa_t **lv_obj_get_style_opa_layered**(const lv_obj_t *obj, *lv_part_t* part)

static inline const lv_color_filter_dsc_t ***lv_obj_get_style_color_filter_dsc**(const lv_obj_t *obj, *lv_part_t* part)

static inline lv_opa_t **lv_obj_get_style_color_filter_opa**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_color_t* **lv_obj_get_style_recolor**(const lv_obj_t *obj, *lv_part_t* part)

static inline lv_opa_t **lv_obj_get_style_recolor_opa**(const lv_obj_t *obj, *lv_part_t* part)

static inline const lv_anim_t ***lv_obj_get_style_anim**(const lv_obj_t *obj, *lv_part_t* part)

static inline uint32_t **lv_obj_get_style_anim_duration**(const lv_obj_t *obj, *lv_part_t* part)

static inline const *lv_style_transition_dsc_t* ***lv_obj_get_style_transition**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_blend_mode_t* **lv_obj_get_style_blend_mode**(const lv_obj_t *obj, *lv_part_t* part)

static inline uint16_t **lv_obj_get_style_layout**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_base_dir_t* **lv_obj_get_style_base_dir**(const lv_obj_t *obj, *lv_part_t* part)

static inline const void ***lv_obj_get_style_bitmap_mask_src**(const lv_obj_t *obj, *lv_part_t* part)

static inline uint32_t **lv_obj_get_style_rotary_sensitivity**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_flex_flow_t* **lv_obj_get_style_flex_flow**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_flex_align_t* **lv_obj_get_style_flex_main_place**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_flex_align_t* **lv_obj_get_style_flex_cross_place**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_flex_align_t* **lv_obj_get_style_flex_track_place**(const lv_obj_t *obj, *lv_part_t* part)

static inline uint8_t **lv_obj_get_style_flex_grow**(const lv_obj_t *obj, *lv_part_t* part)

static inline const int32_t ***lv_obj_get_style_grid_column_dsc_array**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_grid_align_t* **lv_obj_get_style_grid_column_align**(const lv_obj_t *obj, *lv_part_t* part)

static inline const int32_t ***lv_obj_get_style_grid_row_dsc_array**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_grid_align_t* **lv_obj_get_style_grid_row_align**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_grid_cell_column_pos**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_grid_align_t* **lv_obj_get_style_grid_cell_x_align**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_grid_cell_column_span**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_grid_cell_row_pos**(const lv_obj_t *obj, *lv_part_t* part)

static inline *lv_grid_align_t* **lv_obj_get_style_grid_cell_y_align**(const lv_obj_t *obj, *lv_part_t* part)

static inline int32_t **lv_obj_get_style_grid_cell_row_span**(const lv_obj_t *obj, *lv_part_t* part)

void **lv_obj_set_style_width**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_min_width**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_max_width**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_height**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_min_height**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_max_height**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_length**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_x**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_y**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_align**(lv_obj_t *obj, *lv_align_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_transform_width**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_transform_height**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_translate_x**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_translate_y**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_translate_radial**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_transform_scale_x**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_transform_scale_y**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_transform_rotation**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_transform_pivot_x**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_transform_pivot_y**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_transform_skew_x**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_transform_skew_y**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_pad_top**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_pad_bottom**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_pad_left**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_pad_right**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_pad_row**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_pad_column**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_pad_radial**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_margin_top**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_margin_bottom**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_margin_left**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_margin_right**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_bg_color**(lv_obj_t *obj, *lv_color_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_bg_opa**(lv_obj_t *obj, lv_opa_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_bg_grad_color**(lv_obj_t *obj, *lv_color_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_bg_grad_dir**(lv_obj_t *obj, *lv_grad_dir_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_bg_main_stop**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_bg_grad_stop**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_bg_main_opa**(lv_obj_t *obj, lv_opa_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_bg_grad_opa**(lv_obj_t *obj, lv_opa_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_bg_grad**(lv_obj_t *obj, const *lv_grad_dsc_t* *value, *lv_style_selector_t* selector)

void **lv_obj_set_style_bg_image_src**(lv_obj_t *obj, const void *value, *lv_style_selector_t* selector)

void **lv_obj_set_style_bg_image_opa**(lv_obj_t *obj, lv_opa_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_bg_image_recolor**(lv_obj_t *obj, *lv_color_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_bg_image_recolor_opa**(lv_obj_t *obj, lv_opa_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_bg_image_tiled**(lv_obj_t *obj, bool value, *lv_style_selector_t* selector)

void **lv_obj_set_style_border_color**(lv_obj_t *obj, *lv_color_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_border_opa**(lv_obj_t *obj, lv_opa_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_border_width**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_border_side**(lv_obj_t *obj, *lv_border_side_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_border_post**(lv_obj_t *obj, bool value, *lv_style_selector_t* selector)

void **lv_obj_set_style_outline_width**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_outline_color**(lv_obj_t *obj, *lv_color_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_outline_opa**(lv_obj_t *obj, lv_opa_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_outline_pad**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_shadow_width**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_shadow_offset_x**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_shadow_offset_y**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_shadow_spread**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_shadow_color**(lv_obj_t *obj, *lv_color_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_shadow_opa**(lv_obj_t *obj, lv_opa_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_image_opa**(lv_obj_t *obj, lv_opa_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_image_recolor**(lv_obj_t *obj, *lv_color_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_image_recolor_opa**(lv_obj_t *obj, lv_opa_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_line_width**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_line_dash_width**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_line_dash_gap**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_line_rounded**(lv_obj_t *obj, bool value, *lv_style_selector_t* selector)

void **lv_obj_set_style_line_color**(lv_obj_t *obj, *lv_color_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_line_opa**(lv_obj_t *obj, lv_opa_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_arc_width**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_arc_rounded**(lv_obj_t *obj, bool value, *lv_style_selector_t* selector)

void **lv_obj_set_style_arc_color**(lv_obj_t *obj, *lv_color_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_arc_opa**(lv_obj_t *obj, lv_opa_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_arc_image_src**(lv_obj_t *obj, const void *value, *lv_style_selector_t* selector)

void **lv_obj_set_style_text_color**(lv_obj_t *obj, *lv_color_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_text_opa**(lv_obj_t *obj, lv_opa_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_text_font**(lv_obj_t *obj, const lv_font_t *value, *lv_style_selector_t* selector)

void **lv_obj_set_style_text_letter_space**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_text_line_space**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_text_decor**(lv_obj_t *obj, *lv_text_decor_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_text_align**(lv_obj_t *obj, *lv_text_align_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_text_outline_stroke_color**(lv_obj_t *obj, *lv_color_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_text_outline_stroke_width**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_text_outline_stroke_opa**(lv_obj_t *obj, lv_opa_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_radius**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_radial_offset**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_clip_corner**(lv_obj_t *obj, bool value, *lv_style_selector_t* selector)

void **lv_obj_set_style_opa**(lv_obj_t *obj, lv_opa_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_opa_layered**(lv_obj_t *obj, lv_opa_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_color_filter_dsc**(lv_obj_t *obj, const lv_color_filter_dsc_t *value, *lv_style_selector_t* selector)

void **lv_obj_set_style_color_filter_opa**(lv_obj_t *obj, lv_opa_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_recolor**(lv_obj_t *obj, *lv_color_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_recolor_opa**(lv_obj_t *obj, lv_opa_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_anim**(lv_obj_t *obj, const lv_anim_t *value, *lv_style_selector_t* selector)

void **lv_obj_set_style_anim_duration**(lv_obj_t *obj, uint32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_transition**(lv_obj_t *obj, const *lv_style_transition_dsc_t* *value, *lv_style_selector_t* selector)

void **lv_obj_set_style_blend_mode**(lv_obj_t *obj, *lv_blend_mode_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_layout**(lv_obj_t *obj, uint16_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_base_dir**(lv_obj_t *obj, *lv_base_dir_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_bitmap_mask_src**(lv_obj_t *obj, const void *value, *lv_style_selector_t* selector)

void **lv_obj_set_style_rotary_sensitivity**(lv_obj_t *obj, uint32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_flex_flow**(lv_obj_t *obj, *lv_flex_flow_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_flex_main_place**(lv_obj_t *obj, *lv_flex_align_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_flex_cross_place**(lv_obj_t *obj, *lv_flex_align_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_flex_track_place**(lv_obj_t *obj, *lv_flex_align_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_flex_grow**(lv_obj_t *obj, uint8_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_grid_column_dsc_array**(lv_obj_t *obj, const int32_t *value, *lv_style_selector_t* selector)

void **lv_obj_set_style_grid_column_align**(lv_obj_t *obj, *lv_grid_align_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_grid_row_dsc_array**(lv_obj_t *obj, const int32_t *value, *lv_style_selector_t* selector)

void **lv_obj_set_style_grid_row_align**(lv_obj_t *obj, *lv_grid_align_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_grid_cell_column_pos**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_grid_cell_x_align**(lv_obj_t *obj, *lv_grid_align_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_grid_cell_column_span**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_grid_cell_row_pos**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

void **lv_obj_set_style_grid_cell_y_align**(lv_obj_t *obj, *lv_grid_align_t* value, *lv_style_selector_t* selector)

void **lv_obj_set_style_grid_cell_row_span**(lv_obj_t *obj, int32_t value, *lv_style_selector_t* selector)

14.10.17 lv_obj_style_private.h

Functions

void **lv_obj_style_init**(void)

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

void **lv_obj_style_deinit**(void)

Deinitialize the object related style manager module. Called by LVGL in `lv_deinit()`

void **lv_obj_style_create_transition**(lv_obj_t *obj, *lv_part_t* part, *lv_state_t* prev_state, *lv_state_t* new_state, const lv_obj_style_transition_dsc_t *tr)

Used internally to create a style transition

Parametri

- **obj** --
- **part** --
- **prev_state** --
- **new_state** --
- **tr** --

lv_style_state_cmp_t **lv_obj_style_state_compare**(lv_obj_t *obj, *lv_state_t* state1, *lv_state_t* state2)

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

Parametri

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

Ritorna

void **lv_obj_update_layer_type**(lv_obj_t *obj)

Update the layer type of a widget bayed on its current styles. The result will be stored in `obj->spec_attr->layer_type`

Parametri

obj -- the object whose layer should be updated

struct **_lv_obj_style_t**

Public Members

const *lv_style_t* ***style**

uint32_t **selector**

uint32_t **is_local**

uint32_t **is_trans**

uint32_t **is_disabled**

struct **_lv_obj_style_transition_dsc_t**

Public Members

uint16_t **time**

uint16_t **delay**

lv_style_selector_t **selector**

lv_style_prop_t **prop**

lv_anim_path_cb_t **path_cb**

void ***user_data**

14.10.18 lv_obj_tree.h

Typedef

typedef *lv_obj_tree_walk_res_t* (***lv_obj_tree_walk_cb_t**)(lv_obj_t*, void*)

Enums

enum **lv_obj_tree_walk_res_t**

Values:

enumerator **LV_OBJ_TREE_WALK_NEXT**

enumerator **LV_OBJ_TREE_WALK_SKIP_CHILDREN**

enumerator **LV_OBJ_TREE_WALK_END**

Functions

void **lv_obj_delete**(lv_obj_t *obj)

Delete an object and all of its children. Also remove the objects from their group and remove all animations (if any). Send `LV_EVENT_DELETED` to deleted objects.

Parametri

obj -- pointer to an object

void **lv_obj_clean**(lv_obj_t *obj)

Delete all children of an object. Also remove the objects from their group and remove all animations (if any). Send `LV_EVENT_DELETED` to deleted objects.

Parametri

obj -- pointer to an object

void **lv_obj_delete_delayed**(lv_obj_t *obj, uint32_t delay_ms)

Delete an object after some delay

Parametri

- **obj** -- pointer to an object
- **delay_ms** -- time to wait before delete in milliseconds

void **lv_obj_delete_anim_completed_cb**(lv_anim_t *a)

A function to be easily used in animation ready callback to delete an object when the animation is ready

Parametri

a -- pointer to the animation

void **lv_obj_delete_async**(lv_obj_t *obj)

Helper function for asynchronously deleting objects. Useful for cases where you can't delete an object directly in an `LV_EVENT_DELETE` handler (i.e. parent).

> **Vedi anche**
>
> *lv_async_call*

Parametri

obj -- object to delete

void **lv_obj_set_parent**(lv_obj_t *obj, lv_obj_t *parent)

Move the parent of an object. The relative coordinates will be kept.

Parametri

- **obj** -- pointer to an object whose parent needs to be changed
- **parent** -- pointer to the new parent

void **lv_obj_swap**(lv_obj_t *obj1, lv_obj_t *obj2)

Swap the positions of two objects. When used in listboxes, it can be used to sort the listbox items.

Parametri

- **obj1** -- pointer to the first object
- **obj2** -- pointer to the second object

void **lv_obj_move_to_index**(lv_obj_t *obj, int32_t index)

moves the object to the given index in its parent. When used in listboxes, it can be used to sort the listbox items.

> **Nota**
>
> to move to the background: lv_obj_move_to_index(obj, 0)

> **Nota**
>
> to move forward (up): lv_obj_move_to_index(obj, lv_obj_get_index(obj) - 1)

Parametri

- **obj** -- pointer to the object to be moved.
- **index** -- new index in parent. -1 to count from the back

lv_obj_t ***lv_obj_get_screen**(const lv_obj_t *obj)

Get the screen of an object

Parametri

obj -- pointer to an object

Ritorna

pointer to the object's screen

lv_display_t ***lv_obj_get_display**(const lv_obj_t *obj)

Get the display of the object

Parametri

obj -- pointer to an object

Ritorna

pointer to the object's display

lv_obj_t ***lv_obj_get_parent**(const lv_obj_t *obj)

Get the parent of an object

Parametri

obj -- pointer to an object

Ritorna

the parent of the object. (NULL if `obj` was a screen)

lv_obj_t ***lv_obj_get_child**(const lv_obj_t *obj, int32_t idx)

Get the child of an object by the child's index.

Parametri

- **obj** -- pointer to an object whose child should be get
- **idx** -- the index of the child. 0: the oldest (firstly created) child 1: the second oldest child count-1: the youngest -1: the youngest -2: the second youngest

Ritorna

pointer to the child or NULL if the index was invalid

lv_obj_t ***lv_obj_get_child_by_type**(const lv_obj_t *obj, int32_t idx, const lv_obj_class_t *class_p)

Get the child of an object by the child's index. Consider the children only with a given type.

Parametri

- **obj** -- pointer to an object whose child should be get
- **idx** -- the index of the child. 0: the oldest (firstly created) child 1: the second oldest child count-1: the youngest -1: the youngest -2: the second youngest
- **class_p** -- the type of the children to check

Ritorna

pointer to the child or NULL if the index was invalid

lv_obj_t ***lv_obj_get_sibling**(const lv_obj_t *obj, int32_t idx)

Return a sibling of an object

Parametri

- **obj** -- pointer to an object whose sibling should be get
- **idx** -- 0: `obj` itself -1: the first older sibling -2: the next older sibling 1: the first younger sibling 2: the next younger sibling etc

Ritorna

pointer to the requested sibling or NULL if there is no such sibling

lv_obj_t ***lv_obj_get_sibling_by_type**(const lv_obj_t *obj, int32_t idx, const lv_obj_class_t *class_p)

Return a sibling of an object. Consider the siblings only with a given type.

Parametri

- **obj** -- pointer to an object whose sibling should be get
- **idx** -- 0: obj itself -1: the first older sibling -2: the next older sibling 1: the first younger sibling 2: the next younger sibling etc
- **class_p** -- the type of the children to check

Ritorna

pointer to the requested sibling or NULL if there is no such sibling

uint32_t **lv_obj_get_child_count**(const lv_obj_t *obj)

Get the number of children

Parametri

obj -- pointer to an object

Ritorna

the number of children

uint32_t **lv_obj_get_child_count_by_type**(const lv_obj_t *obj, const lv_obj_class_t *class_p)

Get the number of children having a given type.

Parametri

- **obj** -- pointer to an object
- **class_p** -- the type of the children to check

Ritorna

the number of children

void **lv_obj_set_name**(lv_obj_t *obj, const char *name)

Set a name for a widget. The name will be allocated and freed when the widget is deleted or a new name is set.

> **Nota**
>
> If the name ends with a #, older siblings with the same name will be counted, and the # will be replaced by the index of the given widget. For example, creating multiple widgets with the name "mybtn_#" will result in resolved names like "mybtn_0", "mybtn_1", "mybtn_2", etc. The name is resolved when `lv_obj_get_name_resolved` is called, so the result reflects the currently existing widgets at that time.

Parametri

- **obj** -- pointer to an object
- **name** -- the name to set. If set to `NULL` the default "<widget_type>_#" name will be used.

void **lv_obj_set_name_static**(lv_obj_t *obj, const char *name)

Set a name for a widget. Only a pointer will be saved.

> **Nota**
>
> If the name ends with a #, older siblings with the same name will be counted, and the # will be replaced by the index of the given widget. For example, creating multiple widgets with the name "mybtn_#" will result in resolved names like "mybtn_0", "mybtn_1", "mybtn_2", etc. The name is resolved when `lv_obj_get_name_resolved` is called, so the result reflects the currently existing widgets at that time.

Parametri

- **obj** -- pointer to an object
- **name** -- the name to set. If set to `NULL` the default "<widget_type>_#" name will be used.

const char ***lv_obj_get_name**(const lv_obj_t *obj)

Get the set name as it was set.

Parametri

obj -- pointer to an object

Ritorna
get the set name or NULL if it wasn't set yet

void **lv_obj_get_name_resolved**(const lv_obj_t *obj, char buf[], size_t buf_size)

Get the set name or craft a name automatically.

> **Nota**
>
> If the name ends with a #, older siblings with the same name will be counted, and the # will be replaced by the index of the given widget. For example, creating multiple widgets with the name "mybtn_#" will result in resolved names like "mybtn_0", "mybtn_1", "mybtn_2", etc. The name is resolved when `lv_obj_get_name_resolved` is called, so the result reflects the currently existing widgets at that time.

Parametri
- **obj** -- pointer to an object
- **buf** -- buffer to store the name
- **buf_size** -- the size of the buffer in bytes

lv_obj_t ***lv_obj_find_by_name**(const lv_obj_t *parent, const char *name)

Find a child with a given name on a parent. This child doesn't have to be the direct child of the parent. First direct children of the parent will be checked, and the direct children of the first child, etc. (Breadth-first search).

If the name of a widget was not set a name like "lv_button_1" will be created for it using `lv_obj_get_name_resolved`.

Parametri
parent -- the widget where the search should start

Ritorna
the found widget or NULL if not found.

lv_obj_t ***lv_obj_get_child_by_name**(const lv_obj_t *parent, const char *name_path)

Get an object by name. The name can be a path too, for example "main_container/lv_button_1/label". In this case the first part of the name-path should be the direct child of the parent, the second part, should the direct child of first one, etc.

If the name of a widget was not set a name like "lv_button_1" will be created for it using `lv_obj_get_name_resolved`.

Parametri
parent -- the widget where the search should start

Ritorna
the found widget or NULL if not found.

int32_t **lv_obj_get_index**(const lv_obj_t *obj)

Get the index of a child.

Parametri
obj -- pointer to an object

Ritorna
the child index of the object. E.g. 0: the oldest (firstly created child). (-1 if child could not be found or no parent exists)

int32_t **lv_obj_get_index_by_type**(const lv_obj_t *obj, const lv_obj_class_t *class_p)

Get the index of a child. Consider the children only with a given type.

Parametri
- **obj** -- pointer to an object
- **class_p** -- the type of the children to check

Ritorna
the child index of the object. E.g. 0: the oldest (firstly created child with the given class). (-1 if child could not be found or no parent exists)

void **lv_obj_tree_walk**(lv_obj_t *start_obj, *lv_obj_tree_walk_cb_t* cb, void *user_data)

Iterate through all children of any object.

Parametri

- **start_obj** -- start integrating from this object
- **cb** -- call this callback on the objects
- **user_data** -- pointer to any user related data (will be passed to `cb`)

void **lv_obj_dump_tree**(lv_obj_t *start_obj)

Iterate through all children of any object and print their ID.

Parametri

start_obj -- start integrating from this object

14.10.19 lv_refr.h

Functions

void **lv_refr_now**(lv_display_t *disp)

Redraw the invalidated areas now. Normally the redrawing is periodically executed in `lv_timer_handler` but a long blocking process can prevent the call of `lv_timer_handler`. In this case if the GUI is updated in the process (e.g. progress bar) this function can be called when the screen should be updated.

Parametri

disp -- pointer to display to refresh. NULL to refresh all displays.

void **lv_obj_redraw**(lv_layer_t *layer, lv_obj_t *obj)

Redrawn on object and all its children using the passed draw context

Parametri

- **layer** -- pointer to a layer where to draw.
- **obj** -- the start object from the redraw should start

void **lv_display_refr_timer**(lv_timer_t *timer)

Called periodically to handle the refreshing

Parametri

timer -- pointer to the timer itself, or `NULL`

14.10.20 lv_refr_private.h

Functions

void **lv_refr_init**(void)

Initialize the screen refresh subsystem

void **lv_refr_deinit**(void)

Deinitialize the screen refresh subsystem

void **lv_inv_area**(lv_display_t *disp, const *lv_area_t* *area_p)

Invalidate an area on display to redraw it

Parametri

- **area_p** -- pointer to area which should be invalidated (NULL: delete the invalidated areas)
- **disp** -- pointer to display where the area should be invalidated (NULL can be used if there is only one display)

lv_display_t ***lv_refr_get_disp_refreshing**(void)

Get the display which is being refreshed

Ritorna

the display being refreshed

void **lv_refr_set_disp_refreshing**(lv_display_t *disp)

Set the display which is being refreshed

Parametri

disp -- the display being refreshed

lv_obj_t ***lv_refr_get_top_obj**(const *lv_area_t* *area_p, lv_obj_t *obj)

Search the most top object which fully covers an area

Parametri

- **area_p** -- pointer to an area
- **obj** -- the first object to start the searching (typically a screen)

Ritorna

void **lv_obj_refr**(lv_layer_t *layer, lv_obj_t *obj)

Render an object to a layer

Parametri

- **layer** -- target drawing layer
- **obj** -- object to render

14.11 display

14.11.1 lv_display.h

Defines

LV_HOR_RES

The horizontal resolution of the currently active display.

LV_VER_RES

The vertical resolution of the currently active display.

LV_DPX_CALC(dpi, n)

See `lv_dpx()` and `lv_display_dpx()`. Same as Android's DIP. (Different name is chosen to avoid mistype between LV_DPI and LV_DIP)

- 40 dip is 40 px on a 160 DPI screen (distance = 1/4 inch).
- 40 dip is 80 px on a 320 DPI screen (distance still = 1/4 inch).

> **Vedi anche**
>
> https://stackoverflow.com/questions/2025282/what-is-the-difference-between-px-dip-dp-and-sp

LV_DPX(n)

Typedef

typedef void (***lv_display_flush_cb_t**)(lv_display_t *disp, const *lv_area_t* *area, uint8_t *px_map)

typedef void (***lv_display_flush_wait_cb_t**)(lv_display_t *disp)

Enums

enum **lv_display_rotation_t**

Values:

enumerator **LV_DISPLAY_ROTATION_0**

enumerator **LV_DISPLAY_ROTATION_90**

enumerator **LV_DISPLAY_ROTATION_180**

enumerator **LV_DISPLAY_ROTATION_270**

enum **lv_display_render_mode_t**

Values:

enumerator **LV_DISPLAY_RENDER_MODE_PARTIAL**

Use the buffer(s) to render the screen is smaller parts. This way the buffers can be smaller then the display to save RAM. At least 1/10 screen size buffer(s) are recommended.

enumerator **LV_DISPLAY_RENDER_MODE_DIRECT**

The buffer(s) has to be screen sized and LVGL will render into the correct location of the buffer. This way the buffer always contain the whole image. Only the changed ares will be updated. With 2 buffers the buffers' content are kept in sync automatically and in flush_cb only address change is required.

enumerator **LV_DISPLAY_RENDER_MODE_FULL**

Always redraw the whole screen even if only one pixel has been changed. With 2 buffers in flush_cb only an address change is required.

enum **lv_screen_load_anim_t**

Values:

enumerator **LV_SCREEN_LOAD_ANIM_NONE**

enumerator **LV_SCREEN_LOAD_ANIM_OVER_LEFT**

enumerator **LV_SCREEN_LOAD_ANIM_OVER_RIGHT**

enumerator **LV_SCREEN_LOAD_ANIM_OVER_TOP**

enumerator **LV_SCREEN_LOAD_ANIM_OVER_BOTTOM**

enumerator **LV_SCREEN_LOAD_ANIM_MOVE_LEFT**

enumerator **LV_SCREEN_LOAD_ANIM_MOVE_RIGHT**

enumerator **LV_SCREEN_LOAD_ANIM_MOVE_TOP**

enumerator **LV_SCREEN_LOAD_ANIM_MOVE_BOTTOM**

enumerator **LV_SCREEN_LOAD_ANIM_FADE_IN**

enumerator **LV_SCREEN_LOAD_ANIM_FADE_ON**

enumerator **LV_SCREEN_LOAD_ANIM_FADE_OUT**

enumerator **LV_SCREEN_LOAD_ANIM_OUT_LEFT**

enumerator **LV_SCREEN_LOAD_ANIM_OUT_RIGHT**

enumerator **LV_SCREEN_LOAD_ANIM_OUT_TOP**

enumerator **LV_SCREEN_LOAD_ANIM_OUT_BOTTOM**

Functions

lv_display_t ***lv_display_create**(int32_t hor_res, int32_t ver_res)

Create a new display with the given resolution

Parametri

- **hor_res** -- horizontal resolution in pixels
- **ver_res** -- vertical resolution in pixels

Ritorna

pointer to a display object or `NULL` on error

void **lv_display_delete**(lv_display_t *disp)

Remove a display

Parametri

disp -- pointer to display

void **lv_display_set_default**(lv_display_t *disp)

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

Parametri

disp -- pointer to a display

lv_display_t ***lv_display_get_default**(void)

Get the default display

Ritorna

pointer to the default display

lv_display_t ***lv_display_get_next**(lv_display_t *disp)

Get the next display.

Parametri

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

Ritorna

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

void **lv_display_set_resolution**(lv_display_t *disp, int32_t hor_res, int32_t ver_res)

Sets the resolution of a display. `LV_EVENT_RESOLUTION_CHANGED` event will be sent. Here the native resolution of the device should be set. If the display will be rotated later with `lv_display_set_rotation` LVGL will swap the hor. and ver. resolution automatically.

Parametri

- **disp** -- pointer to a display
- **hor_res** -- the new horizontal resolution
- **ver_res** -- the new vertical resolution

void **lv_display_set_physical_resolution**(lv_display_t *disp, int32_t hor_res, int32_t ver_res)

It's not mandatory to use the whole display for LVGL, however in some cases physical resolution is important. For example the touchpad still sees whole resolution and the values needs to be converted to the active LVGL display area.

Parametri

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

- **hor_res** -- the new physical horizontal resolution, or -1 to assume it's the same as the normal hor. res.
- **ver_res** -- the new physical vertical resolution, or -1 to assume it's the same as the normal hor. res.

void **lv_display_set_offset**(lv_display_t *disp, int32_t x, int32_t y)

If physical resolution is not the same as the normal resolution the offset of the active display area can be set here.

Parametri

- **disp** -- pointer to a display
- **x** -- X offset
- **y** -- Y offset

void **lv_display_set_rotation**(lv_display_t *disp, *lv_display_rotation_t* rotation)

Set the rotation of this display. LVGL will swap the horizontal and vertical resolutions internally.

Parametri

- **disp** -- pointer to a display (NULL to use the default display)
- **rotation** -- `LV_DISPLAY_ROTATION_0/90/180/270`

void **lv_display_set_matrix_rotation**(lv_display_t *disp, bool enable)

Use matrix rotation for the display. This function is depended on `LV_DRAW_TRANSFORM_USE_MATRIX`

Parametri

- **disp** -- pointer to a display (NULL to use the default display)
- **enable** -- true: enable matrix rotation, false: disable

void **lv_display_set_dpi**(lv_display_t *disp, int32_t dpi)

Set the DPI (dot per inch) of the display. dpi = sqrt(hor_res^2 + ver_res^2) / diagonal"

Parametri

- **disp** -- pointer to a display
- **dpi** -- the new DPI

int32_t **lv_display_get_horizontal_resolution**(const lv_display_t *disp)

Get the horizontal resolution of a display.

Parametri

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

Ritorna

the horizontal resolution of the display.

int32_t **lv_display_get_vertical_resolution**(const lv_display_t *disp)

Get the vertical resolution of a display

Parametri

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

Ritorna

the vertical resolution of the display

int32_t **lv_display_get_original_horizontal_resolution**(const lv_display_t *disp)

Get the original horizontal resolution of a display without considering rotation

Parametri

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

Ritorna

the horizontal resolution of the display.

int32_t **lv_display_get_original_vertical_resolution**(const lv_display_t *disp)

Get the original vertical resolution of a display without considering rotation

Parametri

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

Ritorna

the vertical resolution of the display

int32_t **lv_display_get_physical_horizontal_resolution**(const lv_display_t *disp)

Get the physical horizontal resolution of a display

Parametri
disp -- pointer to a display (NULL to use the default display)
Ritorna
the physical horizontal resolution of the display

int32_t **lv_display_get_physical_vertical_resolution**(const lv_display_t *disp)

Get the physical vertical resolution of a display
Parametri
disp -- pointer to a display (NULL to use the default display)
Ritorna
the physical vertical resolution of the display

int32_t **lv_display_get_offset_x**(const lv_display_t *disp)

Get the horizontal offset from the full / physical display
Parametri
disp -- pointer to a display (NULL to use the default display)
Ritorna
the horizontal offset from the physical display

int32_t **lv_display_get_offset_y**(const lv_display_t *disp)

Get the vertical offset from the full / physical display
Parametri
disp -- pointer to a display (NULL to use the default display)
Ritorna
the horizontal offset from the physical display

lv_display_rotation_t **lv_display_get_rotation**(lv_display_t *disp)

Get the current rotation of this display.
Parametri
disp -- pointer to a display (NULL to use the default display)
Ritorna
the current rotation

bool **lv_display_get_matrix_rotation**(lv_display_t *disp)

Get if matrix rotation is enabled for a display or not
Parametri
disp -- pointer to a display (NULL to use the default display)
Ritorna
true: matrix rotation is enabled; false: disabled

int32_t **lv_display_get_dpi**(const lv_display_t *disp)

Get the DPI of the display
Parametri
disp -- pointer to a display (NULL to use the default display)
Ritorna
dpi of the display

void **lv_display_set_buffers**(lv_display_t *disp, void *buf1, void *buf2, uint32_t buf_size, *lv_display_render_mode_t* render_mode)

Set the buffers for a display, similarly to `lv_display_set_draw_buffers`, but accept the raw buffer pointers. For DIRECT/FULL rending modes, the buffer size must be at least `hor_res * ver_res * lv_color_format_get_size(lv_display_get_color_format(disp))`
Parametri
- **disp** -- pointer to a display
- **buf1** -- first buffer
- **buf2** -- second buffer (can be `NULL`)
- **buf_size** -- buffer size in byte
- **render_mode** -- LV_DISPLAY_RENDER_MODE_PARTIAL/DIRECT/FULL

void **lv_display_set_buffers_with_stride**(lv_display_t *disp, void *buf1, void *buf2, uint32_t buf_size, uint32_t stride, *lv_display_render_mode_t* render_mode)

Set the frame buffers for a display, similarly to `lv_display_set_buffers`, but allow for a custom stride as required by a display controller. This allows the frame buffers to have a stride alignment different from the rest of the buffers `@param disp pointer to a display @param buf1 first buffer @param buf2 second buffer (can be` NULL`)

Parametri

- **buf_size** -- buffer size in byte
- **stride** -- buffer stride in bytes
- **render_mode** -- LV_DISPLAY_RENDER_MODE_PARTIAL/DIRECT/FULL

void **lv_display_set_draw_buffers**(lv_display_t *disp, lv_draw_buf_t *buf1, lv_draw_buf_t *buf2)

Set the buffers for a display, accept a draw buffer pointer. Normally use `lv_display_set_buffers` is enough for most cases. Use this function when an existing lv_draw_buf_t is available.

Parametri

- **disp** -- pointer to a display
- **buf1** -- first buffer
- **buf2** -- second buffer (can be `NULL`)

void **lv_display_set_3rd_draw_buffer**(lv_display_t *disp, lv_draw_buf_t *buf3)

Set the third draw buffer for a display.

Parametri

- **disp** -- pointer to a display
- **buf3** -- third buffer

void **lv_display_set_render_mode**(lv_display_t *disp, *lv_display_render_mode_t* render_mode)

Set display render mode

Parametri

- **disp** -- pointer to a display
- **render_mode** -- LV_DISPLAY_RENDER_MODE_PARTIAL/DIRECT/FULL

void **lv_display_set_flush_cb**(lv_display_t *disp, *lv_display_flush_cb_t* flush_cb)

Set the flush callback which will be called to copy the rendered image to the display.

Parametri

- **disp** -- pointer to a display
- **flush_cb** -- the flush callback (`px_map` contains the rendered image as raw pixel map and it should be copied to `area` on the display)

void **lv_display_set_flush_wait_cb**(lv_display_t *disp, *lv_display_flush_wait_cb_t* wait_cb)

Set a callback to be used while LVGL is waiting flushing to be finished. It can do any complex logic to wait, including semaphores, mutexes, polling flags, etc. If not set the `disp->flushing` flag is used which can be cleared with *`lv_display_flush_ready()`*

Parametri

- **disp** -- pointer to a display
- **wait_cb** -- a callback to call while LVGL is waiting for flush ready. If NULL *`lv_display_flush_ready()`* can be used to signal that flushing is ready.

void **lv_display_set_color_format**(lv_display_t *disp, *lv_color_format_t* color_format)

Set the color format of the display.

> **Nota**
>
> To change the endianness of the rendered image in case of RGB565 format (i.e. swap the 2 bytes) call `lv_draw_sw_rgb565_swap` in the flush_cb

Parametri

- **disp** -- pointer to a display
- **color_format** -- Possible values are
  - LV_COLOR_FORMAT_RGB565
  - LV_COLOR_FORMAT_RGB888
  - LV_COLOR_FORMAT_XRGB888
  - LV_COLOR_FORMAT_ARGB888

lv_color_format_t **lv_display_get_color_format**(lv_display_t *disp)

Get the color format of the display

Parametri

disp -- pointer to a display

Ritorna

the color format

void **lv_display_set_tile_cnt**(lv_display_t *disp, uint32_t tile_cnt)

Set the number of tiles for parallel rendering.

Parametri

- **disp** -- pointer to a display
- **tile_cnt** -- number of tiles (1 =< tile_cnt < 256)

uint32_t **lv_display_get_tile_cnt**(lv_display_t *disp)

Get the number of tiles used for parallel rendering

Parametri

disp -- pointer to a display

Ritorna

number of tiles

void **lv_display_set_antialiasing**(lv_display_t *disp, bool en)

Enable anti-aliasing for the render engine

Parametri

- **disp** -- pointer to a display
- **en** -- true/false

bool **lv_display_get_antialiasing**(lv_display_t *disp)

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

Parametri

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

Ritorna

true/false

void **lv_display_flush_ready**(lv_display_t *disp)

Call from the display driver when the flushing is finished

Parametri

disp -- pointer to display whose `flush_cb` was called

bool **lv_display_flush_is_last**(lv_display_t *disp)

Tell if it's the last area of the refreshing process. Can be called from `flush_cb` to execute some special display refreshing if needed when all areas area flushed.

Parametri

disp -- pointer to display

Ritorna

true: it's the last area to flush; false: there are other areas too which will be refreshed soon

bool **lv_display_is_double_buffered**(lv_display_t *disp)

lv_obj_t ***lv_display_get_screen_active**(lv_display_t *disp)

Return a pointer to the active screen on a display

Parametri

disp -- pointer to display which active screen should be get. (NULL to use the default screen)

Ritorna

pointer to the active screen object (loaded by '*lv_screen_load()*')

lv_obj_t ***lv_display_get_screen_prev**(lv_display_t *disp)

Return with a pointer to the previous screen. Only used during screen transitions.

Parametri

disp -- pointer to display which previous screen should be get. (NULL to use the default screen)

Ritorna

pointer to the previous screen object or NULL if not used now

lv_obj_t ***lv_display_get_layer_top**(lv_display_t *disp)

Return the top layer. The top layer is the same on all screens and it is above the normal screen layer.

Parametri

disp -- pointer to display which top layer should be get. (NULL to use the default screen)

Ritorna

pointer to the top layer object

lv_obj_t ***lv_display_get_layer_sys**(lv_display_t *disp)

Return the sys. layer. The system layer is the same on all screen and it is above the normal screen and the top layer.

Parametri

disp -- pointer to display which sys. layer should be retrieved. (NULL to use the default screen)

Ritorna

pointer to the sys layer object

lv_obj_t ***lv_display_get_layer_bottom**(lv_display_t *disp)

Return the bottom layer. The bottom layer is the same on all screen and it is under the normal screen layer. It's visible only if the screen is transparent.

Parametri

disp -- pointer to display (NULL to use the default screen)

Ritorna

pointer to the bottom layer object

lv_obj_t ***lv_display_get_screen_by_name**(const lv_display_t *disp, const char *screen_name)

Get screen by its name on a display. The name should be set by `lv_obj_set_name()` or `lv_obj_set_name_static()`.

Parametri

- **disp** -- pointer to a display or NULL to use default display
- **screen_name** -- name of the screen to get

Ritorna

pointer to the screen, or NULL if not found.

void **lv_screen_load**(struct *_lv_obj_t* *scr)

Load a screen on the default display

Parametri

scr -- pointer to a screen

void **lv_screen_load_anim**(lv_obj_t *scr, *lv_screen_load_anim_t* anim_type, uint32_t time, uint32_t delay, bool auto_del)

Switch screen with animation

Parametri

- **scr** -- pointer to the new screen to load
- **anim_type** -- type of the animation from `lv_screen_load_anim_t`, e.g. `LV_SCREEN_LOAD_ANIM_MOVE_LEFT`
- **time** -- time of the animation

- **delay** -- delay before the transition
- **auto_del** -- true: automatically delete the old screen

lv_obj_t ***lv_screen_active**(void)

Get the active screen of the default display

Ritorna

pointer to the active screen

lv_obj_t ***lv_layer_top**(void)

Get the top layer of the default display

Ritorna

pointer to the top layer

lv_obj_t ***lv_layer_sys**(void)

Get the system layer of the default display

Ritorna

pointer to the sys layer

lv_obj_t ***lv_layer_bottom**(void)

Get the bottom layer of the default display

Ritorna

pointer to the bottom layer

void **lv_display_add_event_cb**(lv_display_t *disp, *lv_event_cb_t* event_cb, *lv_event_code_t* filter, void *user_data)

Add an event handler to the display

Parametri

- **disp** -- pointer to a display
- **event_cb** -- an event callback
- **filter** -- event code to react or `LV_EVENT_ALL`
- **user_data** -- optional user_data

uint32_t **lv_display_get_event_count**(lv_display_t *disp)

Get the number of event attached to a display

Parametri

disp -- pointer to a display

Ritorna

number of events

lv_event_dsc_t ***lv_display_get_event_dsc**(lv_display_t *disp, uint32_t index)

Get an event descriptor for an event

Parametri

- **disp** -- pointer to a display
- **index** -- the index of the event

Ritorna

the event descriptor

bool **lv_display_delete_event**(lv_display_t *disp, uint32_t index)

Remove an event

Parametri

- **disp** -- pointer to a display
- **index** -- the index of the event to remove

Ritorna

true: and event was removed; false: no event was removed

uint32_t **lv_display_remove_event_cb_with_user_data**(lv_display_t *disp, *lv_event_cb_t* event_cb, void *user_data)

Remove an event_cb with user_data

Parametri

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

- **event_cb** -- the event_cb of the event to remove
- **user_data** -- user_data

Ritorna
the count of the event removed

lv_result_t **lv_display_send_event**(lv_display_t *disp, *lv_event_code_t* code, void *param)

Send an event to a display

Parametri
- **disp** -- pointer to a display
- **code** -- an event code. LV_EVENT_...
- **param** -- optional param

Ritorna
LV_RESULT_OK: disp wasn't deleted in the event.

lv_area_t ***lv_event_get_invalidated_area**(lv_event_t *e)

Get the area to be invalidated. Can be used in `LV_EVENT_INVALIDATE_AREA`

Parametri
e -- pointer to an event

Ritorna
the area to invalidated (can be modified as required)

void **lv_display_set_theme**(lv_display_t *disp, lv_theme_t *th)

Set the theme of a display. If there are no user created widgets yet the screens' theme will be updated

Parametri
- **disp** -- pointer to a display
- **th** -- pointer to a theme

lv_theme_t ***lv_display_get_theme**(lv_display_t *disp)

Get the theme of a display

Parametri
disp -- pointer to a display

Ritorna
the display's theme (can be NULL)

uint32_t **lv_display_get_inactive_time**(const lv_display_t *disp)

Get elapsed time since last user activity on a display (e.g. click)

Parametri
disp -- pointer to a display (NULL to get the overall smallest inactivity)

Ritorna
elapsed ticks (milliseconds) since the last activity

void **lv_display_trigger_activity**(lv_display_t *disp)

Manually trigger an activity on a display

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

void **lv_display_enable_invalidation**(lv_display_t *disp, bool en)

Temporarily enable and disable the invalidation of the display.

Parametri
- **disp** -- pointer to a display (NULL to use the default display)
- **en** -- true: enable invalidation; false: invalidation

bool **lv_display_is_invalidation_enabled**(lv_display_t *disp)

Get display invalidation is enabled.

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

Ritorna
return true if invalidation is enabled

lv_timer_t ***lv_display_get_refr_timer**(lv_display_t *disp)

Get a pointer to the screen refresher timer to modify its parameters with `lv_timer_...` functions.

Parametri
disp -- pointer to a display
Ritorna
pointer to the display refresher timer. (NULL on error)

void **lv_display_delete_refr_timer**(lv_display_t *disp)

Delete screen refresher timer
Parametri
disp -- pointer to a display

bool **lv_display_register_vsync_event**(lv_display_t *disp, *lv_event_cb_t* event_cb, void *user_data)

Register vsync event of a display. `LV_EVENT_VSYNC` event will be sent periodically. Please don't use it in display event listeners, as it may cause memory leaks and illegal access issues.
Parametri
- **disp** -- pointer to a display
- **event_cb** -- an event callback
- **user_data** -- optional user_data

bool **lv_display_unregister_vsync_event**(lv_display_t *disp, *lv_event_cb_t* event_cb, void *user_data)

Unregister vsync event of a display. `LV_EVENT_VSYNC` event won't be sent periodically. Please don't use it in display event listeners, as it may cause memory leaks and illegal access issues.
Parametri
- **disp** -- pointer to a display
- **event_cb** -- an event callback
- **user_data** -- optional user_data

lv_result_t **lv_display_send_vsync_event**(lv_display_t *disp, void *param)

Send an vsync event to a display
Parametri
- **disp** -- pointer to a display
- **param** -- optional param

Ritorna
LV_RESULT_OK: disp wasn't deleted in the event.

void **lv_display_set_user_data**(lv_display_t *disp, void *user_data)

void **lv_display_set_driver_data**(lv_display_t *disp, void *driver_data)

void ***lv_display_get_user_data**(lv_display_t *disp)

void ***lv_display_get_driver_data**(lv_display_t *disp)

lv_draw_buf_t ***lv_display_get_buf_active**(lv_display_t *disp)

void **lv_display_rotate_area**(lv_display_t *disp, *lv_area_t* *area)

Rotate an area in-place according to the display's rotation
Parametri
- **disp** -- pointer to a display
- **area** -- pointer to an area to rotate

uint32_t **lv_display_get_draw_buf_size**(lv_display_t *disp)

Get the size of the draw buffers
Parametri
disp -- pointer to a display
Ritorna
the size of the draw buffer in bytes for valid display, 0 otherwise

uint32_t **lv_display_get_invalidated_draw_buf_size**(lv_display_t *disp, uint32_t width, uint32_t height)

Get the size of the invalidated draw buffer. Can be used in the flush callback to get the number of bytes used in the current render buffer.

Parametri
- **disp** -- pointer to a display
- **width** -- the width of the invalidated area
- **height** -- the height of the invalidated area

Ritorna
the size of the invalidated draw buffer in bytes, not accounting for any preceding palette information for a valid display, 0 otherwise

int32_t **lv_dpx**(int32_t n)

For default display, computes the number of pixels (a distance or size) as if the display had 160 DPI. This allows you to specify 1/160-th fractions of an inch to get real distance on the display that will be consistent regardless of its current DPI. It ensures `lv_dpx(100)`, for example, will have the same physical size regardless to the DPI of the display.

Parametri
n -- number of 1/160-th-inch units to compute with

Ritorna
number of pixels to use to make that distance

int32_t **lv_display_dpx**(const lv_display_t *disp, int32_t n)

For specified display, computes the number of pixels (a distance or size) as if the display had 160 DPI. This allows you to specify 1/160-th fractions of an inch to get real distance on the display that will be consistent regardless of its current DPI. It ensures `lv_dpx(100)`, for example, will have the same physical size regardless to the DPI of the display.

Parametri
- **disp** -- pointer to display whose dpi should be considered
- **n** -- number of 1/160-th-inch units to compute with

Ritorna
number of pixels to use to make that distance

14.11.2 lv_display_private.h

Defines

LV_INV_BUF_SIZE

Buffer size for invalid areas

struct **_lv_display_t**

Public Members

int32_t **hor_res**

Horizontal resolution.

int32_t **ver_res**

Vertical resolution.

int32_t **physical_hor_res**

Horizontal resolution of the full / physical display. Set to -1 for fullscreen mode.

int32_t **physical_ver_res**

Vertical resolution of the full / physical display. Set to -1 for fullscreen mode.

int32_t **offset_x**

Horizontal offset from the full / physical display. Set to 0 for fullscreen mode.

int32_t **offset_y**

Vertical offset from the full / physical display. Set to 0 for fullscreen mode.

uint32_t **dpi**

DPI (dot per inch) of the display. Default value is `LV_DPI_DEF`.

lv_draw_buf_t ***buf_1**

lv_draw_buf_t ***buf_2**

lv_draw_buf_t ***buf_3**

lv_draw_buf_t ***buf_act**

Internal, used by the library

lv_display_flush_cb_t **flush_cb**

MANDATORY: Write the internal buffer (draw_buf) to the display. '*lv_display_flush_ready()*' has to be called when finished

lv_display_flush_wait_cb_t **flush_wait_cb**

Used to wait while flushing is ready. It can do any complex logic to wait, including semaphores, mutexes, polling flags, etc. If not set `flushing` flag is used which can be cleared with *`lv_display_flush_ready()`*

volatile int **flushing**

1: flushing is in progress. (It can't be a bit field because when it's cleared from IRQ Read-Modify-Write issue might occur)

volatile int **flushing_last**

1: It was the last chunk to flush. (It can't be a bit field because when it's cleared from IRQ Read-Modify-Write issue might occur)

volatile uint32_t **last_area**

1: last area is being rendered

volatile uint32_t **last_part**

1: last part of the current area is being rendered

lv_display_render_mode_t **render_mode**

uint32_t **antialiasing**

1: anti-aliasing is enabled on this display.

uint32_t **tile_cnt**

Divide the display buffer into these number of tiles

uint32_t **stride_is_auto**

1: The stride of the buffers was not set explicitly.

uint32_t **rendering_in_progress**

1: The current screen rendering is in progress

lv_color_format_t **color_format**

lv_area_t **inv_areas**[32]

Invalidated (marked to redraw) areas

uint8_t **inv_area_joined**[32]

uint32_t **inv_p**

int32_t **inv_en_cnt**

lv_ll_t **sync_areas**

Double buffer sync areas (redrawn during last refresh)

lv_draw_buf_t **_static_buf1**

Used when user pass in a raw buffer as display draw buffer

lv_draw_buf_t **_static_buf2**

lv_layer_t ***layer_head**

void (***layer_init**)(lv_display_t *disp, lv_layer_t *layer)

void (***layer_deinit**)(lv_display_t *disp, lv_layer_t *layer)

lv_obj_t ****screens**

Screens of the display Array of screen objects.

lv_obj_t ***sys_layer**

> **Vedi anche**
>
> *lv_display_get_layer_sys*

lv_obj_t ***top_layer**

> **Vedi anche**
>
> *lv_display_get_layer_top*

lv_obj_t ***act_scr**

Currently active screen on this display

lv_obj_t ***bottom_layer**

> **Vedi anche**
>
> *lv_display_get_layer_bottom*

lv_obj_t ***prev_scr**

Previous screen. Used during screen animations

lv_obj_t ***scr_to_load**

The screen prepared to load in lv_screen_load_anim

uint32_t **screen_cnt**

uint8_t **draw_prev_over_act**

uint8_t **del_prev**

1: Draw previous screen over active screen

void ***driver_data**

1: Automatically delete the previous screen when the screen load animation is ready Custom user data

void ***user_data**

Custom user data

lv_event_list_t **event_list**

uint32_t **rotation**

Element of lv_display_rotation_t

uint32_t **matrix_rotation**

1: Use matrix for display rotation

lv_theme_t ***theme**

The theme assigned to the screen

lv_timer_t ***refr_timer**

A timer which periodically checks the dirty areas and refreshes them

uint32_t **last_activity_time**

Last time when there was activity on this display

lv_area_t **refreshed_area**

The area being refreshed

uint32_t **vsync_count**

lv_obj_t *`perf_label`

lv_sysmon_backend_data_t `perf_sysmon_backend`

lv_sysmon_perf_info_t `perf_sysmon_info`

lv_obj_t *`mem_label`

14.12 draw

14.12.1 lv_draw.h

Defines

`LV_DRAW_UNIT_NONE`

Modified by NXP in 2024

`LV_DRAW_UNIT_IDLE`

The draw unit is idle, new dispatching might be requested to try again

Enums

enum `lv_draw_task_type_t`

Values:

enumerator `LV_DRAW_TASK_TYPE_NONE`

enumerator `LV_DRAW_TASK_TYPE_FILL`

enumerator `LV_DRAW_TASK_TYPE_BORDER`

enumerator `LV_DRAW_TASK_TYPE_BOX_SHADOW`

enumerator `LV_DRAW_TASK_TYPE_LETTER`

enumerator `LV_DRAW_TASK_TYPE_LABEL`

enumerator `LV_DRAW_TASK_TYPE_IMAGE`

enumerator `LV_DRAW_TASK_TYPE_LAYER`

enumerator `LV_DRAW_TASK_TYPE_LINE`

enumerator `LV_DRAW_TASK_TYPE_ARC`

enumerator `LV_DRAW_TASK_TYPE_TRIANGLE`

enumerator **LV_DRAW_TASK_TYPE_MASK_RECTANGLE**

enumerator **LV_DRAW_TASK_TYPE_MASK_BITMAP**

enumerator **LV_DRAW_TASK_TYPE_VECTOR**

enumerator **LV_DRAW_TASK_TYPE_3D**

enum **lv_draw_task_state_t**

Values:

enumerator **LV_DRAW_TASK_STATE_BLOCKED**

Waiting for an other task to be finished. For example in case of `LV_DRAW_TASK_TYPE_LAYER` (used to blend a layer) is blocked until all the draw tasks of the layer is rendered.

enumerator **LV_DRAW_TASK_STATE_WAITING**

The draw task is added to the layers list and waits to be rendered.

enumerator **LV_DRAW_TASK_STATE_QUEUED**

The draw task is added to the command queue of the draw unit. As the queued task are executed in order it's possible to queue multiple draw task (for the same draw unit) even if they are depending on each other. Therefore `lv_draw_get_available_task` and `lv_draw_get_next_available_task` can return draw task for the same draw unit even if a dependent draw task is not finished ready yet.

enumerator **LV_DRAW_TASK_STATE_IN_PROGRESS**

The draw task is being rendered. This draw task needs to be finished before `lv_draw_get_available_task` and `lv_draw_get_next_available_task` would return any depending draw tasks.

enumerator **LV_DRAW_TASK_STATE_FINISHED**

The draw task is rendered. It will be removed from the draw task list of the layer and freed automatically.

Functions

void **lv_draw_init**(void)

Used internally to initialize the drawing module

void **lv_draw_deinit**(void)

Deinitialize the drawing module

void ***lv_draw_create_unit**(size_t size)

Allocate a new draw unit with the given size and appends it to the list of draw units

Parametri

size -- the size to allocate. E.g. `sizeof(my_draw_unit_t)`, where the first element of `my_draw_unit_t` is `lv_draw_unit_t`.

lv_draw_task_t ***lv_draw_add_task**(lv_layer_t *layer, const *lv_area_t* *coords, *lv_draw_task_type_t* type)

Add an empty draw task to the draw task list of a layer.

Parametri

- **layer** -- pointer to a layer
- **coords** -- the coordinates of the draw task

Ritorna

the created draw task which needs to be further configured e.g. by added a draw descriptor

void **lv_draw_finalize_task_creation**(lv_layer_t *layer, lv_draw_task_t *t)

Needs to be called when a draw task is created and configured. It will send an event about the new draw task to the widget and assign it to a draw unit.

Parametri

- **layer** -- pointer to a layer
- **t** -- pointer to a draw task

void **lv_draw_dispatch**(void)

Try dispatching draw tasks to draw units

bool **lv_draw_dispatch_layer**(lv_display_t *disp, lv_layer_t *layer)

Used internally to try dispatching draw tasks of a specific layer

Parametri

- **disp** -- pointer to a display on which the dispatching was requested
- **layer** -- pointer to a layer

Ritorna

at least one draw task is being rendered (maybe it was taken earlier)

void **lv_draw_dispatch_wait_for_request**(void)

Wait for a new dispatch request. It's blocking if `LV_USE_OS == 0` else it yields

void **lv_draw_wait_for_finish**(void)

Wait for draw finish in case of asynchronous task execution. If `LV_USE_OS == 0` it just return.

void **lv_draw_dispatch_request**(void)

When a draw unit finished a draw task it needs to request dispatching to let LVGL assign a new draw task to it

uint32_t **lv_draw_get_unit_count**(void)

Get the total number of draw units.

lv_draw_task_t ***lv_draw_get_available_task**(lv_layer_t *layer, lv_draw_task_t *t_prev, uint8_t draw_unit_id)

If there is only one draw unit check the first draw task if it's available. If there are multiple draw units call `lv_draw_get_next_available_task` to find a task.

Parametri

- **layer** -- the draw layer to search in
- **t_prev** -- continue searching from this task
- **draw_unit_id** -- check the task where `preferred_draw_unit_id` equals this value or `LV_DRAW_UNIT_NONE`

Ritorna

an available draw task or NULL if there is not any

lv_draw_task_t ***lv_draw_get_next_available_task**(lv_layer_t *layer, lv_draw_task_t *t_prev, uint8_t draw_unit_id)

Find and available draw task

Parametri

- **layer** -- the draw layer to search in
- **t_prev** -- continue searching from this task
- **draw_unit_id** -- check the task where `preferred_draw_unit_id` equals this value or `LV_DRAW_UNIT_NONE`

Ritorna

an available draw task or NULL if there is not any

uint32_t **lv_draw_get_dependent_count**(lv_draw_task_t *t_check)

Tell how many draw task are waiting to be drawn on the area of `t_check`. It can be used to determine if a GPU shall combine many draw tasks into one or not. If a lot of tasks are waiting for the current ones it makes sense to draw them one-by-one to not block the dependent tasks' rendering

Parametri

t_check -- the task whose dependent tasks shall be counted

Ritorna
number of tasks depending on `t_check`

void **`lv_layer_init`**(lv_layer_t *layer)

Initialize a layer

Parametri
`layer` -- pointer to a layer to initialize

void **`lv_layer_reset`**(lv_layer_t *layer)

Reset the layer to a drawable state

Parametri
`layer` -- pointer to a layer to reset

lv_layer_t ***`lv_draw_layer_create`**(lv_layer_t *parent_layer, *lv_color_format_t* color_format, const *lv_area_t* *area)

Create (allocate) a new layer on a parent layer

Parametri
- **`parent_layer`** -- the parent layer to which the layer will be merged when it's rendered
- **`color_format`** -- the color format of the layer
- **`area`** -- the areas of the layer (absolute coordinates)

Ritorna
the new target_layer or NULL on error

void **`lv_draw_layer_init`**(lv_layer_t *layer, lv_layer_t *parent_layer, *lv_color_format_t* color_format, const *lv_area_t* *area)

Initialize a layer which is allocated by the user

Parametri
- **`layer`** -- pointer the layer to initialize (its lifetime needs to be managed by the user)
- **`parent_layer`** -- the parent layer to which the layer will be merged when it's rendered
- **`color_format`** -- the color format of the layer
- **`area`** -- the areas of the layer (absolute coordinates)

Ritorna
the new target_layer or NULL on error

void ***`lv_draw_layer_alloc_buf`**(lv_layer_t *layer)

Try to allocate a buffer for the layer.

Parametri
`layer` -- pointer to a layer

Ritorna
pointer to the allocated aligned buffer or NULL on failure

void ***`lv_draw_layer_go_to_xy`**(lv_layer_t *layer, int32_t x, int32_t y)

Got to a pixel at X and Y coordinate on a layer

Parametri
- **`layer`** -- pointer to a layer
- **`x`** -- the target X coordinate
- **`y`** -- the target X coordinate

Ritorna
`buf` offset to point to the given X and Y coordinate

lv_draw_task_type_t **`lv_draw_task_get_type`**(const lv_draw_task_t *t)

Get the type of a draw task

Parametri
`t` -- the draw task to get the type of

Ritorna
the draw task type

void *`lv_draw_task_get_draw_dsc`(const lv_draw_task_t *t)

Get the draw descriptor of a draw task

Parametri

t -- the draw task to get the draw descriptor of

Ritorna

a void pointer to the draw descriptor

void `lv_draw_task_get_area`(const lv_draw_task_t *t, *lv_area_t* *area)

Get the draw area of a draw task

Parametri

- **t** -- the draw task to get the draw area of
- **area** -- the destination where the draw area will be stored

struct `_lv_layer_t`

Public Members

lv_draw_buf_t ***draw_buf**

Target draw buffer of the layer

lv_draw_task_t ***draw_task_head**

Linked list of draw tasks

lv_layer_t ***parent**

Parent layer

lv_layer_t ***next**

Next layer

void ***user_data**

User data

lv_area_t **buf_area**

The absolute coordinates of the buffer

lv_area_t **phy_clip_area**

The physical clipping area relative to the display

lv_area_t **_clip_area**

NEVER USE IT DRAW UNITS. USED INTERNALLY DURING DRAW TASK CREATION. The current clip area with absolute coordinates, always the same or smaller than `buf_area` Can be set before new draw tasks are added to indicate the clip area of the draw tasks. Therefore `lv_draw_add_task()` always saves it in the new draw task to know the clip area when the draw task was added. During drawing the draw units also sees the saved clip_area and should use it during drawing. During drawing the layer's clip area shouldn't be used as it might be already changed for other draw tasks.

int32_t **partial_y_offset**

Partial y offset

lv_color32_t **recolor**

Recolor of the layer

lv_color_format_t **color_format**

The color format of the layer. LV_COLOR_FORMAT_...

bool **all_tasks_added**

Flag indicating all tasks are added

lv_opa_t **opa**

Opacity of the layer

struct **lv_draw_dsc_base_t**

Public Members

lv_obj_t ***obj**

The widget for which draw descriptor was created

uint32_t **part**

The widget part for which draw descriptor was created

uint32_t **id1**

A widget type specific ID (e.g. table row index). See the docs of the given widget.

uint32_t **id2**

A widget type specific ID (e.g. table column index). See the docs of the given widget.

lv_layer_t ***layer**

The target layer

size_t **dsc_size**

Size of the specific draw descriptor into which this base descriptor is embedded

void ***user_data**

Any custom user data

14.12.2 lv_draw_3d.h

Functions

void **lv_draw_3d_dsc_init**(*lv_draw_3d_dsc_t* *dsc)

Initialize a 3D draw descriptor

Parametri

dsc -- pointer to a draw descriptor

lv_draw_3d_dsc_t ***lv_draw_task_get_3d_dsc**(lv_draw_task_t *task)

Try to get a 3D draw descriptor from a draw task.

Parametri

task -- draw task

Ritorna

the task's draw descriptor or NULL if the task is not of type LV_DRAW_TASK_TYPE_3D

void **lv_draw_3d**(lv_layer_t *layer, const *lv_draw_3d_dsc_t* *dsc, const *lv_area_t* *coords)

Create a 3D draw task

Parametri

- **layer** -- pointer to a layer
- **dsc** -- pointer to an initialized `lv_draw_3d_dsc_t` variable

struct **lv_draw_3d_dsc_t**

Public Members

lv_draw_dsc_base_t **base**

lv_3dtexture_id_t **tex_id**

lv_opa_t **opa**

14.12.3 lv_draw_arc.h

Functions

void **lv_draw_arc_dsc_init**(*lv_draw_arc_dsc_t* *dsc)

Initialize an arc draw descriptor.

Parametri

dsc -- pointer to a draw descriptor

lv_draw_arc_dsc_t ***lv_draw_task_get_arc_dsc**(lv_draw_task_t *task)

Try to get an arc draw descriptor from a draw task.

Parametri

task -- draw task

Ritorna

the task's draw descriptor or NULL if the task is not of type LV_DRAW_TASK_TYPE_ARC

void **lv_draw_arc**(lv_layer_t *layer, const *lv_draw_arc_dsc_t* *dsc)

Create an arc draw task.

Parametri

- **layer** -- pointer to a layer
- **dsc** -- pointer to an initialized draw descriptor variable

void **lv_draw_arc_get_area**(int32_t x, int32_t y, uint16_t radius, lv_value_precise_t start_angle, lv_value_precise_t end_angle, int32_t w, bool rounded, *lv_area_t* *area)

Get an area the should be invalidated when the arcs angle changed between start_angle and end_ange

Parametri

- **x** -- the x coordinate of the center of the arc
- **y** -- the y coordinate of the center of the arc
- **radius** -- the radius of the arc
- **start_angle** -- the start angle of the arc (0 deg on the bottom, 90 deg on the right)
- **end_angle** -- the end angle of the arc
- **w** -- width of the arc
- **rounded** -- true: the arc is rounded
- **area** -- store the area to invalidate here

struct **lv_draw_arc_dsc_t**

Public Members

lv_draw_dsc_base_t **base**

lv_color_t **color**

The color of the arc

int32_t **width**

The width (thickness) of the arc

lv_value_precise_t **start_angle**

The start angle in 1 degree units (if `LV_USE_FLOAT` is enabled a float number can be also used) 0° is the 3 o'clock position, 90° is the 6 o'clock, etc.

lv_value_precise_t **end_angle**

The end angle, similarly to start_angle.

lv_point_t **center**

The center point of the arc.

const void ***img_src**

An image source to be used instead of `color`. `NULL` if unused

uint16_t **radius**

The outer radius of the arc

lv_opa_t **opa**

Opacity of the arc in 0...255 range. LV_OPA_TRANSP, LV_OPA_10, LV_OPA_20, .. LV_OPA_COVER can be used as well

uint8_t **rounded**

1: Make the arc ends rounded

14.12.4 lv_draw_buf.h

Defines

LV_STRIDE_AUTO

Use this value to let LVGL calculate stride automatically

LV_DRAW_BUF_STRIDE(w, cf)

Stride alignment for draw buffers. It may vary between different color formats and hardware. Refine it to suit your needs.

LV_DRAW_BUF_SIZE(w, h, cf)

Allocate a slightly larger buffer, so we can adjust the start address to meet alignment

LV_DRAW_BUF_DEFINE_STATIC(name, _w, _h, _cf)

Define a static draw buffer with the given width, height, and color format. Stride alignment is set to LV_DRAW_BUF_STRIDE_ALIGN.

For platform that needs special buffer alignment, call LV_DRAW_BUF_INIT_STATIC.

LV_DRAW_BUF_INIT_STATIC(name)

Typedef

typedef void *(*`lv_draw_buf_malloc_cb`)(size_t size, *lv_color_format_t* color_format)

typedef void (*`lv_draw_buf_free_cb`)(void *draw_buf)

typedef void *(*`lv_draw_buf_align_cb`)(void *buf, *lv_color_format_t* color_format)

typedef void (*`lv_draw_buf_cache_operation_cb`)(const lv_draw_buf_t *draw_buf, const *lv_area_t* *area)

typedef uint32_t (*`lv_draw_buf_width_to_stride_cb`)(uint32_t w, *lv_color_format_t* color_format)

Functions

void `lv_draw_buf_init_with_default_handlers`(lv_draw_buf_handlers_t *handlers)

Initialize the draw buffer with the default handlers.

Parametri

`handlers` -- the draw buffer handlers to set

void `lv_draw_buf_handlers_init`(lv_draw_buf_handlers_t *handlers, *lv_draw_buf_malloc_cb* buf_malloc_cb, *lv_draw_buf_free_cb* buf_free_cb, *lv_draw_buf_align_cb* align_pointer_cb, *lv_draw_buf_cache_operation_cb* invalidate_cache_cb, *lv_draw_buf_cache_operation_cb* flush_cache_cb, *lv_draw_buf_width_to_stride_cb* width_to_stride_cb)

Initialize the draw buffer with given handlers.

Parametri

- `handlers` -- the draw buffer handlers to set
- `buf_malloc_cb` -- the callback to allocate memory for the buffer
- `buf_free_cb` -- the callback to free memory of the buffer
- `align_pointer_cb` -- the callback to align the buffer
- `invalidate_cache_cb` -- the callback to invalidate the cache of the buffer
- `flush_cache_cb` -- the callback to flush buffer
- `width_to_stride_cb` -- the callback to calculate the stride based on the width and color format

lv_draw_buf_handlers_t *`lv_draw_buf_get_handlers`(void)

Get the struct which holds the callbacks for draw buf management. Custom callback can be set on the returned value

Ritorna

pointer to the struct of handlers

lv_draw_buf_handlers_t *`lv_draw_buf_get_font_handlers`(void)

lv_draw_buf_handlers_t *`lv_draw_buf_get_image_handlers`(void)

void *`lv_draw_buf_align`(void *buf, *lv_color_format_t* color_format)

Align the address of a buffer. The buffer needs to be large enough for the real data after alignment

Parametri

- `buf` -- the data to align
- `color_format` -- the color format of the buffer

Ritorna

the aligned buffer

void *`lv_draw_buf_align_ex`(const lv_draw_buf_handlers_t *handlers, void *buf, *lv_color_format_t* color_format)

Align the address of a buffer with custom draw buffer handlers. The buffer needs to be large enough for the real data after alignment

Parametri

- **handlers** -- the draw buffer handlers
- **buf** -- the data to align
- **color_format** -- the color format of the buffer

Ritorna

the aligned buffer

void `lv_draw_buf_invalidate_cache`(const lv_draw_buf_t *draw_buf, const *lv_area_t* *area)

Invalidate the cache of the buffer

Parametri

- **draw_buf** -- the draw buffer needs to be invalidated
- **area** -- the area to invalidate in the buffer, use NULL to invalidate the whole draw buffer address range

void `lv_draw_buf_flush_cache`(const lv_draw_buf_t *draw_buf, const *lv_area_t* *area)

Flush the cache of the buffer

Parametri

- **draw_buf** -- the draw buffer needs to be flushed
- **area** -- the area to flush in the buffer, use NULL to flush the whole draw buffer address range

uint32_t `lv_draw_buf_width_to_stride`(uint32_t w, *lv_color_format_t* color_format)

Calculate the stride in bytes based on a width and color format

Parametri

- **w** -- the width in pixels
- **color_format** -- the color format

Ritorna

the stride in bytes

uint32_t `lv_draw_buf_width_to_stride_ex`(const lv_draw_buf_handlers_t *handlers, uint32_t w, *lv_color_format_t* color_format)

Calculate the stride in bytes based on a width and color format

Parametri

- **handlers** -- the draw buffer handlers
- **w** -- the width in pixels
- **color_format** -- the color format

Ritorna

the stride in bytes

void `lv_draw_buf_clear`(lv_draw_buf_t *draw_buf, const *lv_area_t* *a)

Clear an area on the buffer

Parametri

- **draw_buf** -- pointer to draw buffer
- **a** -- the area to clear, or NULL to clear the whole buffer

void `lv_draw_buf_copy`(lv_draw_buf_t *dest, const *lv_area_t* *dest_area, const lv_draw_buf_t *src, const *lv_area_t* *src_area)

Copy an area from a buffer to another

> **Nota**
>
> `dest_area` and `src_area` should have the same width and height

> **Nota**
>
> `dest` and `src` should have same color format. Color converting is not supported fow now.

Parametri

- **`dest`** -- pointer to the destination draw buffer
- **`dest_area`** -- the area to copy from the destination buffer, if NULL, use the whole buffer
- **`src`** -- pointer to the source draw buffer
- **`src_area`** -- the area to copy from the destination buffer, if NULL, use the whole buffer

lv_draw_buf_t *`lv_draw_buf_create`(uint32_t w, uint32_t h, *lv_color_format_t* cf, uint32_t stride)

Note: Eventually, lv_draw_buf_malloc/free will be kept as private. For now, we use `create` to distinguish with malloc.

Create an draw buf by allocating struct for `lv_draw_buf_t` and allocating a buffer for it that meets specified requirements.

Parametri

- **`w`** -- the buffer width in pixels
- **`h`** -- the buffer height in pixels
- **`cf`** -- the color format for image
- **`stride`** -- the stride in bytes for image. Use 0 for automatic calculation based on w, cf, and global stride alignment configuration.

lv_draw_buf_t *`lv_draw_buf_create_ex`(const lv_draw_buf_handlers_t *handlers, uint32_t w, uint32_t h, *lv_color_format_t* cf, uint32_t stride)

Note: Eventually, lv_draw_buf_malloc/free will be kept as private. For now, we use `create` to distinguish with malloc.

Create an draw buf by allocating struct for `lv_draw_buf_t` and allocating a buffer for it that meets specified requirements.

Parametri

- **`handlers`** -- the draw buffer handlers
- **`w`** -- the buffer width in pixels
- **`h`** -- the buffer height in pixels
- **`cf`** -- the color format for image
- **`stride`** -- the stride in bytes for image. Use 0 for automatic calculation based on w, cf, and global stride alignment configuration.

lv_draw_buf_t *`lv_draw_buf_dup`(const lv_draw_buf_t *draw_buf)

Duplicate a draw buf with same image size, stride and color format. Copy the image data too.

Parametri

`draw_buf` -- the draw buf to duplicate

Ritorna

the duplicated draw buf on success, NULL if failed

lv_draw_buf_t *`lv_draw_buf_dup_ex`(const lv_draw_buf_handlers_t *handlers, const lv_draw_buf_t *draw_buf)

Duplicate a draw buf with same image size, stride and color format. Copy the image data too.

Parametri

- **`handlers`** -- the draw buffer handlers
- **`draw_buf`** -- the draw buf to duplicate

Ritorna

the duplicated draw buf on success, NULL if failed

lv_result_t `lv_draw_buf_init`(lv_draw_buf_t *draw_buf, uint32_t w, uint32_t h, *lv_color_format_t* cf, uint32_t stride, void *data, uint32_t data_size)

Initialize a draw buf with the given buffer and parameters. Clear draw buffer flag to zero.

Parametri

- **draw_buf** -- the draw buf to initialize
- **w** -- the buffer width in pixels
- **h** -- the buffer height in pixels
- **cf** -- the color format
- **stride** -- the stride in bytes. Use 0 for automatic calculation
- **data** -- the buffer used for drawing. Unaligned `data` will be aligned internally
- **data_size** -- the size of the buffer in bytes

Ritorna

return LV_RESULT_OK on success, LV_RESULT_INVALID otherwise

lv_draw_buf_t ***lv_draw_buf_reshape**(lv_draw_buf_t *draw_buf, *lv_color_format_t* cf, uint32_t w, uint32_t h, uint32_t stride)

Keep using the existing memory, reshape the draw buffer to the given width and height. Return NULL if data_size is smaller than the required size.

Parametri

- **draw_buf** -- pointer to a draw buffer
- **cf** -- the new color format, use 0 or LV_COLOR_FORMAT_UNKNOWN to keep using the original color format.
- **w** -- the new width in pixels
- **h** -- the new height in pixels
- **stride** -- the stride in bytes for image. Use 0 for automatic calculation.

void **lv_draw_buf_destroy**(lv_draw_buf_t *draw_buf)

Destroy a draw buf by freeing the actual buffer if it's marked as LV_IMAGE_FLAGS_ALLOCATED in header. Then free the lv_draw_buf_t struct.

Parametri

draw_buf -- the draw buffer to destroy

void ***lv_draw_buf_goto_xy**(const lv_draw_buf_t *buf, uint32_t x, uint32_t y)

Return pointer to the buffer at the given coordinates

lv_result_t **lv_draw_buf_adjust_stride**(lv_draw_buf_t *src, uint32_t stride)

Adjust the stride of a draw buf in place.

Parametri

- **src** -- pointer to a draw buffer
- **stride** -- the new stride in bytes for image. Use LV_STRIDE_AUTO for automatic calculation.

Ritorna

LV_RESULT_OK: success or LV_RESULT_INVALID: failed

lv_result_t **lv_draw_buf_premultiply**(lv_draw_buf_t *draw_buf)

Premultiply draw buffer color with alpha channel. If it's already premultiplied, return directly. Only color formats with alpha channel will be processed.

Ritorna

LV_RESULT_OK: premultiply success

bool **lv_draw_buf_has_flag**(const lv_draw_buf_t *draw_buf, *lv_image_flags_t* flag)

void **lv_draw_buf_set_flag**(lv_draw_buf_t *draw_buf, *lv_image_flags_t* flag)

void **lv_draw_buf_clear_flag**(lv_draw_buf_t *draw_buf, *lv_image_flags_t* flag)

lv_result_t **lv_draw_buf_from_image**(lv_draw_buf_t *buf, const *lv_image_dsc_t* *img)

As of now, draw buf share same definition as `lv_image_dsc_t`. And is interchangeable with `lv_image_dsc_t`.

void **lv_draw_buf_to_image**(const lv_draw_buf_t *buf, *lv_image_dsc_t* *img)

void **lv_draw_buf_set_palette**(lv_draw_buf_t *draw_buf, uint8_t index, *lv_color32_t* color)

Set the palette color of an indexed image. Valid only for `LV_COLOR_FORMAT_I1/2/4/8`

Parametri

- **draw_buf** -- pointer to an image descriptor
- **index** -- the palette color to set:
  - for `LV_COLOR_FORMAT_I1`: 0..1
  - for `LV_COLOR_FORMAT_I2`: 0..3
  - for `LV_COLOR_FORMAT_I4`: 0..15
  - for `LV_COLOR_FORMAT_I8`: 0..255
- **color** -- the color to set in *lv_color32_t* format

void **lv_image_buf_set_palette**(*lv_image_dsc_t* *dsc, uint8_t id, *lv_color32_t* c)

Deprecated:

Use lv_draw_buf_set_palette instead.

void **lv_image_buf_free**(*lv_image_dsc_t* *dsc)

Deprecated:

Use lv_draw_buffer_create/destroy instead. Free the data pointer and dsc struct of an image.

struct **_lv_draw_buf_t**

Public Members

lv_image_header_t **header**

uint32_t **data_size**

Total buf size in bytes

uint8_t ***data**

void ***unaligned_data**

Unaligned address of `data`, used internally by lvgl

const lv_draw_buf_handlers_t ***handlers**

draw buffer alloc/free ops.

14.12.5 lv_draw_buf_private.h

Functions

void **lv_draw_buf_init_handlers**(void)

Called internally to initialize the draw_buf_handlers in lv_global

struct **_lv_draw_buf_handlers_t**

Public Members

lv_draw_buf_malloc_cb **buf_malloc_cb**

lv_draw_buf_free_cb **buf_free_cb**

lv_draw_buf_align_cb **align_pointer_cb**

lv_draw_buf_cache_operation_cb **invalidate_cache_cb**

lv_draw_buf_cache_operation_cb **flush_cache_cb**

lv_draw_buf_width_to_stride_cb **width_to_stride_cb**

14.12.6 lv_draw_image.h

Typedef

typedef void (*`lv_draw_image_core_cb`)(lv_draw_task_t *t, const lv_draw_image_dsc_t *draw_dsc, const lv_image_decoder_dsc_t *decoder_dsc, lv_draw_image_sup_t *sup, const *lv_area_t* *img_coords, const *lv_area_t* *clipped_img_area)

PErform the actual rendering of a decoded image

Param t
pointer to a draw task

Param draw_dsc
the draw descriptor of the image

Param decoder_dsc
pointer to the decoded image's descriptor

Param sup
supplementary data

Param img_coords
the absolute coordinates of the image

Param clipped_img_area
the absolute clip coordinates

Functions

void `lv_draw_image_dsc_init`(lv_draw_image_dsc_t *dsc)

Initialize an image draw descriptor.

Parametri
`dsc` -- pointer to a draw descriptor

lv_draw_image_dsc_t *`lv_draw_task_get_image_dsc`(lv_draw_task_t *task)

Try to get an image draw descriptor from a draw task.

Parametri
`task` -- draw task

Ritorna
the task's draw descriptor or NULL if the task is not of type LV_DRAW_TASK_TYPE_IMAGE

void `lv_draw_image`(lv_layer_t *layer, const lv_draw_image_dsc_t *dsc, const *lv_area_t* *coords)

Create an image draw task

> **Nota**
>
> `coords` can be small than the real image area (if only a part of the image is rendered) or can be larger (in case of tiled images). .

Parametri

- `layer` -- pointer to a layer
- `dsc` -- pointer to an initialized draw descriptor
- `coords` -- the coordinates of the image

void **lv_draw_layer**(lv_layer_t *layer, const lv_draw_image_dsc_t *dsc, const *lv_area_t* *coords)

Create a draw task to blend a layer to another layer

> **Nota**
>
> `coords` can be small than the total widget area from which the layer is created (if only a part of the widget was rendered to a layer)

Parametri

- **layer** -- pointer to a layer
- **dsc** -- pointer to an initialized draw descriptor. `src` must be set to the layer to blend
- **coords** -- the coordinates of the layer.

lv_image_src_t **lv_image_src_get_type**(const void *src)

Get the type of an image source

Parametri

src -- pointer to an image source:

- pointer to an 'lv_image_t' variable (image stored internally and compiled into the code)
- a path to a file (e.g. "S:/folder/image.bin")
- or a symbol (e.g. LV_SYMBOL_CLOSE)

Ritorna

type of the image source LV_IMAGE_SRC_VARIABLE/FILE/SYMBOL/UNKNOWN

struct **_lv_draw_image_dsc_t**

Public Members

lv_draw_dsc_base_t **base**

const void ***src**

The image source: pointer to `lv_image_dsc_t` or a path to a file

lv_image_header_t **header**

The header of the image. Initialized internally in `lv_draw_image`

int32_t **clip_radius**

Clip the corner of the image with this radius. Use `LV_RADIUS_CIRCLE` for max. radius

int32_t **rotation**

The rotation of the image in 0.1 degree unit. E.g. 234 means 23.4°

int32_t **scale_x**

Horizontal scale (zoom) of the image. 256 (LV_SCALE_NONE): means no zoom, 512 double size, 128 half size.

int32_t **scale_y**

Same as `scale_y` but vertically

int32_t **skew_x**

Parallelogram like transformation of the image horizontally in 0.1 degree unit. E.g. 456 means 45.6°.

int32_t **skew_y**

Same as `skew_x` but vertically

lv_point_t **pivot**

The pivot point of transformation (scale and rotation). 0;0 is the top left corner of the image. Can be outside of the image too.

lv_color_t **recolor**

Mix this color to the images. In case of `LV_COLOR_FORMAT_A8` it will be the color of the visible pixels

lv_opa_t **recolor_opa**

The intensity of recoloring. 0 means, no recolor, 255 means full cover (transparent pixels remain transparent)

lv_opa_t **opa**

Opacity in 0...255 range. LV_OPA_TRANSP, LV_OPA_10, LV_OPA_20, .. LV_OPA_COVER can be used as well

lv_blend_mode_t **blend_mode**

Describes how to blend the pixels of the image to the background. See `lv_blend_mode_t` for more details.

uint16_t **antialias**

1: perform the transformation with anti-aliasing

uint16_t **tile**

If the image is smaller than the `image_area` field of `lv_draw_image_dsc_t` tile the image (repeat is both horizontally and vertically) to fill the `image_area` area

lv_draw_image_sup_t ***sup**

Used internally to store some information about the palette or the color of A8 images

lv_area_t **image_area**

Used to indicate the entire original, non-clipped area where the image is to be drawn. This is important for:

i. Layer rendering, where it might happen that only a smaller area of the layer is rendered and e.g. `clip_radius` needs to know what the original image was.
ii. Tiled images, where the target draw area is larger than the image to be tiled.

const *lv_image_dsc_t* ***bitmap_mask_src**

Pointer to an A8 or L8 image descriptor to mask the image with. The mask is always center aligned.

14.12.7 lv_draw_image_private.h

Functions

void **lv_draw_image_normal_helper**(lv_draw_task_t *t, const lv_draw_image_dsc_t *draw_dsc, const *lv_area_t* *coords, *lv_draw_image_core_cb* draw_core_cb)

Can be used by draw units to handle the decoding and prepare everything for the actual image rendering

Parametri

- **t** -- pointer to a draw task

- **draw_dsc** -- the draw descriptor of the image
- **coords** -- the absolute coordinates of the image
- **draw_core_cb** -- a callback to perform the actual rendering

void **lv_draw_image_tiled_helper**(lv_draw_task_t *t, const lv_draw_image_dsc_t *draw_dsc, const *lv_area_t* *coords, *lv_draw_image_core_cb* draw_core_cb)

Can be used by draw units for TILED images to handle the decoding and prepare everything for the actual image rendering

Parametri

- **t** -- pointer to a draw task
- **draw_dsc** -- the draw descriptor of the image
- **coords** -- the absolute coordinates of the image
- **draw_core_cb** -- a callback to perform the actual rendering

void **lv_image_buf_get_transformed_area**(*lv_area_t* *res, int32_t w, int32_t h, int32_t angle, uint16_t scale_x, uint16_t scale_y, const *lv_point_t* *pivot)

Get the area of a rectangle if its rotated and scaled

Parametri

- **res** -- store the coordinates here
- **w** -- width of the rectangle to transform
- **h** -- height of the rectangle to transform
- **angle** -- angle of rotation
- **scale_x** -- zoom in x direction, (256 no zoom)
- **scale_y** -- zoom in y direction, (256 no zoom)
- **pivot** -- x,y pivot coordinates of rotation

struct **_lv_draw_image_sup_t**

Public Members

lv_color_t **alpha_color**

const *lv_color32_t* ***palette**

uint32_t **palette_size**

14.12.8 lv_draw_label.h

Defines

LV_DRAW_LABEL_NO_TXT_SEL

Typedef

typedef void (***lv_draw_glyph_cb_t**)(lv_draw_task_t *t, lv_draw_glyph_dsc_t *dsc, *lv_draw_fill_dsc_t* *fill_dsc, const *lv_area_t* *fill_area)

Passed as a parameter to `lv_draw_label_iterate_characters` to draw the characters one by one

Param t

pointer to a draw task

Param dsc

pointer to `lv_draw_glyph_dsc_t` to describe the character to draw if NULL don't draw character

Param fill_dsc
pointer to a fill descriptor to draw a background for the character or underline or strike through if NULL do not fill anything

Param fill_area
the area to fill if NULL do not fill anything

Functions

void **lv_draw_letter_dsc_init**(*lv_draw_letter_dsc_t* *dsc)

void **lv_draw_label_dsc_init**(*lv_draw_label_dsc_t* *dsc)

Initialize a label draw descriptor

Parametri
dsc -- pointer to a draw descriptor

lv_draw_label_dsc_t ***lv_draw_task_get_label_dsc**(lv_draw_task_t *task)

Try to get a label draw descriptor from a draw task.

Parametri
task -- draw task

Ritorna
the task's draw descriptor or NULL if the task is not of type LV_DRAW_TASK_TYPE_LABEL

void **lv_draw_glyph_dsc_init**(lv_draw_glyph_dsc_t *dsc)

Initialize a glyph draw descriptor. Used internally.

Parametri
dsc -- pointer to a draw descriptor

void **lv_draw_label**(lv_layer_t *layer, const *lv_draw_label_dsc_t* *dsc, const *lv_area_t* *coords)

Create a draw task to render a text

Parametri
- **layer** -- pointer to a layer
- **dsc** -- pointer to draw descriptor
- **coords** -- coordinates of the character

void **lv_draw_character**(lv_layer_t *layer, *lv_draw_label_dsc_t* *dsc, const *lv_point_t* *point, uint32_t unicode_letter)

Create a draw task to render a single character

Parametri
- **layer** -- pointer to a layer
- **dsc** -- pointer to draw descriptor
- **point** -- position of the label
- **unicode_letter** -- the letter to draw

void **lv_draw_letter**(lv_layer_t *layer, *lv_draw_letter_dsc_t* *dsc, const *lv_point_t* *point)

Draw a single letter

Parametri
- **layer** -- pointer to a layer
- **dsc** -- pointer to draw descriptor
- **point** -- position of the label

void **lv_draw_label_iterate_characters**(lv_draw_task_t *t, const *lv_draw_label_dsc_t* *dsc, const *lv_area_t* *coords, *lv_draw_glyph_cb_t* cb)

Should be used during rendering the characters to get the position and other parameters of the characters

Parametri
- **t** -- pointer to a draw task
- **dsc** -- pointer to draw descriptor
- **coords** -- coordinates of the label
- **cb** -- a callback to call to draw each glyphs one by one

void **lv_draw_unit_draw_letter**(lv_draw_task_t *t, lv_draw_glyph_dsc_t *dsc, const *lv_point_t* *pos, const lv_font_t *font, uint32_t letter, *lv_draw_glyph_cb_t* cb)

Draw a single letter using the provided draw unit, glyph descriptor, position, font, and callback.

This function is responsible for rendering a single character from a text string, applying the necessary styling described by the glyph descriptor (`dsc`). It handles the retrieval of the glyph's description, checks its visibility within the clipping area, and invokes the callback (`cb`) to render the glyph at the specified position (`pos`) using the given font (`font`).

Parametri

- **t** -- Pointer to the drawing task.
- **dsc** -- Pointer to the descriptor containing styling for the glyph to be drawn.
- **pos** -- Pointer to the point coordinates where the letter should be drawn.
- **font** -- Pointer to the font containing the glyph.
- **letter** -- The Unicode code point of the letter to be drawn.
- **cb** -- Callback function to execute the actual rendering of the glyph.

struct **lv_draw_label_dsc_t**

Public Members

lv_draw_dsc_base_t **base**

const char ***text**

The text to draw

lv_point_t **text_size**

The size of the text

const lv_font_t ***font**

The font to use. Fallback fonts are also handled.

lv_color_t **color**

Color of the text

int32_t **line_space**

Extra space between the lines

int32_t **letter_space**

Extra space between the characters

int32_t **ofs_x**

Offset the text with this value horizontally

int32_t **ofs_y**

Offset the text with this value vertically

int32_t **rotation**

Rotation of the letters in 0.1 degree unit

uint32_t **sel_start**

The first characters index for selection (not byte index). `LV_DRAW_LABEL_NO_TXT_SEL` for no selection

uint32_t **sel_end**

The last characters's index for selection (not byte index). `LV_DRAW_LABEL_NO_TXT_SEL` for no selection

lv_color_t **sel_color**

Color of the selected characters

lv_color_t **sel_bg_color**

Background color of the selected characters

uint32_t **text_length**

The number of characters to render. 0: means render until reaching the `\0` termination.

lv_text_align_t **align**

The alignment of the text `LV_TEXT_ALIGN_LEFT/RIGHT/CENTER`

lv_base_dir_t **bidi_dir**

The base direction. Used when type setting Right-to-left (e.g. Arabic) texts

lv_opa_t **opa**

Opacity of the text in 0...255 range. LV_OPA_TRANSP, LV_OPA_10, LV_OPA_20, .. LV_OPA_COVER can be used as well

lv_opa_t **outline_stroke_opa**

Letter outline stroke opacity

lv_text_decor_t **decor**

Text decoration, e.g. underline

lv_text_flag_t **flag**

Some flags to control type setting

uint8_t **text_local**

1: malloc a buffer and copy `text` there. 0: `text` will be valid during rendering.

uint8_t **text_static**

Indicate that the text is constant and its pointer can be safely saved e.g. in a cache.

uint8_t **has_bided**

1: already executed lv_bidi_process_paragraph. 0: has not been executed lv_bidi_process_paragraph.

lv_draw_label_hint_t ***hint**

Pointer to an externally stored struct where some data can be cached to speed up rendering

lv_color_t **outline_stroke_color**

int32_t **outline_stroke_width**

struct **lv_draw_letter_dsc_t**

Public Members

lv_draw_dsc_base_t **base**

uint32_t **unicode**

const lv_font_t ***font**

lv_color_t **color**

int32_t **rotation**

int32_t **scale_x**

int32_t **scale_y**

int32_t **skew_x**

int32_t **skew_y**

lv_point_t **pivot**

lv_opa_t **opa**

lv_text_decor_t **decor**

lv_blend_mode_t **blend_mode**

lv_opa_t **outline_stroke_opa**

int32_t **outline_stroke_width**

lv_color_t **outline_stroke_color**

14.12.9 lv_draw_label_private.h

struct **_lv_draw_label_hint_t**

#include <lv_draw_label_private.h> Store some info to speed up drawing of very large texts It takes a lot of time to get the first visible character because all the previous characters needs to be checked to calculate the positions. This structure stores an earlier (e.g. at -1000 px) coordinate and the index of that line. Therefore the calculations can start from here.

Public Members

int32_t **line_start**

Index of the line at `y` coordinate

int32_t **y**

Give the `y` coordinate of the first letter at `line start` index. Relative to the label's coordinates

int32_t **coord_y**

The 'y1' coordinate of the label when the hint was saved. Used to invalidate the hint if the label has moved too much.

struct **_lv_draw_glyph_dsc_t**

Public Members

const void ***glyph_data**

Depends on `format` field, it could be image source or draw buf of bitmap or vector data.

lv_font_glyph_format_t **format**

const *lv_area_t* ***letter_coords**

const *lv_area_t* ***bg_coords**

lv_font_glyph_dsc_t ***g**

lv_color_t **color**

lv_opa_t **opa**

lv_color_t **outline_stroke_color**

lv_opa_t **outline_stroke_opa**

int32_t **outline_stroke_width**

int32_t **rotation**

lv_point_t **pivot**

Rotation pivot point associated with total glyph including line_height

lv_draw_buf_t ***_draw_buf**

a shared draw buf for get_bitmap, do not use it directly, use glyph_data instead

14.12.10 lv_draw_line.h

Functions

void **lv_draw_line_dsc_init**(*lv_draw_line_dsc_t* *dsc)

Initialize a line draw descriptor

Parametri

dsc -- pointer to a draw descriptor

lv_draw_line_dsc_t ***lv_draw_task_get_line_dsc**(lv_draw_task_t *task)

Try to get a line draw descriptor from a draw task.

Parametri

task -- draw task

Ritorna

the task's draw descriptor or NULL if the task is not of type LV_DRAW_TASK_TYPE_LINE

void **lv_draw_line**(lv_layer_t *layer, const *lv_draw_line_dsc_t* *dsc)

Create a line draw task

Parametri

- **layer** -- pointer to a layer
- **dsc** -- pointer to an initialized `lv_draw_line_dsc_t` variable

struct **lv_draw_line_dsc_t**

Public Members

lv_draw_dsc_base_t **base**

lv_point_precise_t **p1**

The first point of the line. If `LV_USE_FLOAT` is enabled float number can be also used

lv_point_precise_t **p2**

The second point of the line. If `LV_USE_FLOAT` is enabled float number can be also used

lv_color_t **color**

The color of the line

int32_t **width**

The width (thickness) of the line

int32_t **dash_width**

The length of a dash (0: don't dash)

int32_t **dash_gap**

The length of the gaps between dashes (0: don't dash)

lv_opa_t **opa**

Opacity of the line in 0...255 range. LV_OPA_TRANSP, LV_OPA_10, LV_OPA_20, .. LV_OPA_COVER can be used as well

uint8_t **round_start**

Make the line start rounded

uint8_t **round_end**

Make the line end rounded

uint8_t **raw_end**

1: Do not bother with line ending (if it's not visible for any reason)

14.12.11 lv_draw_mask.h

Functions

void **lv_draw_mask_rect_dsc_init**(lv_draw_mask_rect_dsc_t *dsc)

Initialize a rectangle mask draw descriptor.

Parametri

dsc -- pointer to a draw descriptor

lv_draw_mask_rect_dsc_t ***lv_draw_task_get_mask_rect_dsc**(lv_draw_task_t *task)

Try to get a rectangle mask draw descriptor from a draw task.

Parametri

task -- draw task

Ritorna

the task's draw descriptor or NULL if the task is not of type LV_DRAW_TASK_TYPE_MASK_RECTANGLE

void **lv_draw_mask_rect**(lv_layer_t *layer, const lv_draw_mask_rect_dsc_t *dsc)

Create a draw task to mask a rectangle from the buffer

Parametri

- **layer** -- pointer to a layer
- **dsc** -- pointer to a draw descriptor

14.12.12 lv_draw_mask_private.h

struct **_lv_draw_mask_rect_dsc_t**

Public Members

lv_draw_dsc_base_t **base**

lv_area_t **area**

The area t mask.

int32_t **radius**

The radius of masking

uint32_t **keep_outside**

0: clear the content out of the `area`. 1: don't touch the area out of `area`

14.12.13 lv_draw_private.h

struct **_lv_draw_task_t**

#include <lv_draw_private.h> Modified by NXP in 2024

Public Members

lv_draw_task_t ***next**

lv_draw_task_type_t **type**

lv_area_t **area**

The area where to draw

lv_area_t **_real_area**

The real draw area. E.g. for shadow, outline, or transformed images it's different from `area`

lv_area_t **clip_area_original**

The original area which is updated

lv_area_t **clip_area**

The clip area of the layer is saved here when the draw task is created. As the clip area of the layer can be changed as new draw tasks are added its current value needs to be saved. Therefore during drawing the layer's clip area shouldn't be used as it might be already changed for other draw tasks.

lv_layer_t ***target_layer**

lv_draw_unit_t ***draw_unit**

volatile int **state**

void ***draw_dsc**

int instead of *lv_draw_task_state_t* to be sure its atomic

lv_opa_t **opa**

Opacity of the layer

uint8_t **preferred_draw_unit_id**

The ID of the draw_unit which should take this task

uint8_t **preference_score**

Set to which extent `preferred_draw_unit_id` is good at this task. 80: means 20% better (faster) than software rendering 100: the default value 110: means 10% worse (slower) than software rendering

struct **_lv_draw_mask_t**

Public Members

void ***user_data**

struct **_lv_draw_unit_t**

Public Members

lv_draw_unit_t ***next**

const char ***name**

Name and ID of the draw unit, for debugging purposes only.

int32_t **idx**

int32_t (*`dispatch_cb`)(lv_draw_unit_t *draw_unit, lv_layer_t *layer)

Called to try to assign a draw task to itself. `lv_draw_get_next_available_task` can be used to get an independent draw task. A draw task should be assign only if the draw unit can draw it too

Param draw_unit
pointer to the draw unit

Param layer
pointer to a layer on which the draw task should be drawn

Return
>=0: The number of taken draw task: 0 means the task has not yet been completed. 1 means a new task has been accepted. -1: The draw unit wanted to work on a task but couldn't do that due to some errors (e.g. out of memory). It signals that LVGL should call the dispatcher later again to let draw unit try to start the rendering again.

int32_t (*`evaluate_cb`)(lv_draw_unit_t *draw_unit, lv_draw_task_t *task)

Param draw_unit

Param task

Return

int32_t (*`wait_for_finish_cb`)(lv_draw_unit_t *draw_unit)

Called to signal the unit to complete all tasks in order to return their ready status. This callback can be implemented in case of asynchronous task processing. Below is an example to show the difference between synchronous and asynchronous:

Synchronous: LVGL thread DRAW thread HW

task1 --> submit --> Receive task1 wait_for_finish() <— task1->state = READY <— Complete task1 task2 --> submit --> Receive task2 wait_for_finish() task2->state = READY <— Complete task2 task3 --> submit --> Receive task3 wait_for_finish() <— task3->state = READY <— Complete task3 task4 --> submit --> Receive task4 wait_for_finish() <— task4->state = READY <— Complete task4 NO MORE TASKS

Asynchronous: LVGL thread DRAW thread HW is IDLE task1 --> queue task1 submit --> Receive task1 task2 --> queue task2 is BUSY (with task1) task3 --> queue task3 still BUSY (with task1) task4 --> queue task4 becomes IDLE <— task1->state = READY <— Complete task1 submit --> Receive task2, task3, task4 NO MORE TASKS *wait_for_finish_cb()* wait_for_finish() <— Complete task2, task3, task4 <— task2->state = READY <— <— task3->state = READY <— <— task4->state = READY <—

Param draw_unit

Return

int32_t (*`delete_cb`)(lv_draw_unit_t *draw_unit)

Called to delete draw unit.

Param draw_unit

Return

struct `lv_draw_global_info_t`

Public Members

lv_draw_unit_t *`unit_head`

uint32_t `unit_cnt`

uint32_t `used_memory_for_layers`

volatile int **dispatch_req**

lv_mutex_t **circle_cache_mutex**

bool **task_running**

14.12.14 lv_draw_rect.h

Defines

LV_RADIUS_CIRCLE

A very big radius to always draw as circle

Functions

void **lv_draw_rect_dsc_init**(*lv_draw_rect_dsc_t* *dsc)

Initialize a rectangle draw descriptor.

Parametri

dsc -- pointer to a draw descriptor

void **lv_draw_fill_dsc_init**(*lv_draw_fill_dsc_t* *dsc)

Initialize a fill draw descriptor.

Parametri

dsc -- pointer to a draw descriptor

lv_draw_fill_dsc_t ***lv_draw_task_get_fill_dsc**(lv_draw_task_t *task)

Try to get a fill draw descriptor from a draw task.

Parametri

task -- draw task

Ritorna

the task's draw descriptor or NULL if the task is not of type LV_DRAW_TASK_TYPE_FILL

void **lv_draw_fill**(lv_layer_t *layer, const *lv_draw_fill_dsc_t* *dsc, const *lv_area_t* *coords)

Fill an area

Parametri

- **layer** -- pointer to a layer
- **dsc** -- pointer to an initialized draw descriptor variable
- **coords** -- the coordinates of the rectangle

void **lv_draw_border_dsc_init**(*lv_draw_border_dsc_t* *dsc)

Initialize a border draw descriptor.

Parametri

dsc -- pointer to a draw descriptor

lv_draw_border_dsc_t ***lv_draw_task_get_border_dsc**(lv_draw_task_t *task)

Try to get a border draw descriptor from a draw task.

Parametri

task -- draw task

Ritorna

the task's draw descriptor or NULL if the task is not of type LV_DRAW_TASK_TYPE_BORDER

void **lv_draw_border**(lv_layer_t *layer, const *lv_draw_border_dsc_t* *dsc, const *lv_area_t* *coords)

Draw a border

Parametri

- **layer** -- pointer to a layer

- **dsc** -- pointer to an initialized draw descriptor variable
- **coords** -- the coordinates of the rectangle

void **lv_draw_box_shadow_dsc_init**(*lv_draw_box_shadow_dsc_t* *dsc)

Initialize a box shadow draw descriptor.

Parametri

dsc -- pointer to a draw descriptor

lv_draw_box_shadow_dsc_t ***lv_draw_task_get_box_shadow_dsc**(lv_draw_task_t *task)

Try to get a box shadow draw descriptor from a draw task.

Parametri

task -- draw task

Ritorna

the task's draw descriptor or NULL if the task is not of type LV_DRAW_TASK_TYPE_BOX_SHADOW

void **lv_draw_box_shadow**(lv_layer_t *layer, const *lv_draw_box_shadow_dsc_t* *dsc, const *lv_area_t* *coords)

Draw a box shadow

Parametri

- **layer** -- pointer to a layer
- **dsc** -- pointer to an initialized draw descriptor variable
- **coords** -- the coordinates of the rectangle

void **lv_draw_rect**(lv_layer_t *layer, const *lv_draw_rect_dsc_t* *dsc, const *lv_area_t* *coords)

The rectangle is a wrapper for fill, border, bg. image and box shadow. Internally fill, border, image and box shadow draw tasks will be created.

Parametri

- **layer** -- pointer to a layer
- **dsc** -- pointer to an initialized draw descriptor variable
- **coords** -- the coordinates of the rectangle

struct **lv_draw_rect_dsc_t**

Public Members

lv_draw_dsc_base_t **base**

int32_t **radius**

const void ***bg_image_src**

const void ***bg_image_symbol_font**

lv_color_t **bg_image_recolor**

lv_opa_t **bg_image_opa**

lv_opa_t **bg_image_recolor_opa**

uint8_t **bg_image_tiled**

lv_opa_t **bg_opa**

lv_opa_t **border_opa**

lv_opa_t **outline_opa**

lv_opa_t **shadow_opa**

lv_color_t **bg_color**

First element of a gradient is a color, so it maps well here

lv_grad_dsc_t **bg_grad**

lv_color_t **border_color**

int32_t **border_width**

lv_border_side_t **border_side**

uint8_t **border_post**

lv_color_t **outline_color**

int32_t **outline_width**

int32_t **outline_pad**

lv_color_t **shadow_color**

int32_t **shadow_width**

int32_t **shadow_offset_x**

int32_t **shadow_offset_y**

int32_t **shadow_spread**

struct **lv_draw_fill_dsc_t**

Public Members

lv_draw_dsc_base_t **base**

int32_t **radius**

Radius, LV_RADIUS_CIRCLE for max. radius

lv_opa_t **opa**

Opacity in 0...255 range. LV_OPA_TRANSP, LV_OPA_10, LV_OPA_20, .. LV_OPA_COVER can be used as well

lv_color_t **color**

The color of the rectangle. If the gradient is set (grad.dir!=LV_GRAD_DIR_NONE) it's ignored.

lv_grad_dsc_t **grad**

Describe a gradient. If `grad.dir` is not `LV_GRAD_DIR_NONE` `color` will be ignored

struct **lv_draw_border_dsc_t**

Public Members

lv_draw_dsc_base_t **base**

int32_t **radius**

Radius, LV_RADIUS_CIRCLE for max. radius

lv_color_t **color**

The color of the border.

int32_t **width**

The width of the border in pixels

lv_opa_t **opa**

Opacity in 0...255 range. LV_OPA_TRANSP, LV_OPA_10, LV_OPA_20, .. LV_OPA_COVER can be used as well

lv_border_side_t **side**

LV_BORDER_SIDE_NONE/LEFT/RIGHT/TOP/BOTTOM/FULL. LV_BORDER_SIDE_INTERNAL is an information for upper layers and shouldn't be used here.

struct **lv_draw_box_shadow_dsc_t**

Public Members

lv_draw_dsc_base_t **base**

int32_t **radius**

Radius, LV_RADIUS_CIRCLE for max. radius

lv_color_t **color**

Color of the the shadow

int32_t **width**

Width of the shadow. (radius of the blur)

int32_t **spread**

Make the rectangle larger with this value in all directions. Can be negative too.

int32_t **ofs_x**

Offset the rectangle horizontally.

int32_t **ofs_y**

Offset the rectangle vertically.

lv_opa_t **opa**

Opacity in 0...255 range. LV_OPA_TRANSP, LV_OPA_10, LV_OPA_20, .. LV_OPA_COVER can be used as well

uint8_t **bg_cover**

Set `bg_cover` to 1 if the background will cover the shadow. It's a hint to the renderer about it might skip some masking.

14.12.15 lv_draw_rect_private.h

14.12.16 lv_draw_triangle.h

Functions

void **lv_draw_triangle_dsc_init**(*lv_draw_triangle_dsc_t* *draw_dsc)

Initialize a triangle draw descriptor

Parametri

draw_dsc -- pointer to a draw descriptor

lv_draw_triangle_dsc_t ***lv_draw_task_get_triangle_dsc**(lv_draw_task_t *task)

Try to get a triangle draw descriptor from a draw task.

Parametri

task -- draw task

Ritorna

the task's draw descriptor or NULL if the task is not of type LV_DRAW_TASK_TYPE_TRIANGLE

void **lv_draw_triangle**(lv_layer_t *layer, const *lv_draw_triangle_dsc_t* *draw_dsc)

Create a triangle draw task

Parametri

- **layer** -- pointer to a layer
- **draw_dsc** -- pointer to an initialized `lv_draw_triangle_dsc_t` object

struct **lv_draw_triangle_dsc_t**

Public Members

lv_draw_dsc_base_t **base**

lv_point_precise_t **p**[3]

Points of the triangle. If `LV_USE_FLOAT` is enabled floats can be used here

lv_color_t **color**

Color of the triangle

lv_opa_t **opa**

Opacity of the arc in 0...255 range. LV_OPA_TRANSP, LV_OPA_10, LV_OPA_20, .. LV_OPA_COVER can be used as well

lv_grad_dsc_t **grad**

Describe a gradient. If `grad.dir` is not `LV_GRAD_DIR_NONE` `color` will be ignored

14.12.17 lv_draw_triangle_private.h

14.12.18 lv_draw_vector.h

Typedef

typedef void (*`vector_draw_task_cb`)(void *ctx, const lv_vector_path_t *path, const lv_vector_draw_dsc_t *dsc)

Enums

enum `lv_vector_fill_t`

Values:

enumerator `LV_VECTOR_FILL_NONZERO`

enumerator `LV_VECTOR_FILL_EVENODD`

enum `lv_vector_stroke_cap_t`

Values:

enumerator `LV_VECTOR_STROKE_CAP_BUTT`

enumerator `LV_VECTOR_STROKE_CAP_SQUARE`

enumerator `LV_VECTOR_STROKE_CAP_ROUND`

enum `lv_vector_stroke_join_t`

Values:

enumerator `LV_VECTOR_STROKE_JOIN_MITER`

enumerator `LV_VECTOR_STROKE_JOIN_BEVEL`

enumerator `LV_VECTOR_STROKE_JOIN_ROUND`

enum `lv_vector_path_quality_t`

Values:

enumerator `LV_VECTOR_PATH_QUALITY_MEDIUM`

enumerator `LV_VECTOR_PATH_QUALITY_HIGH`

enumerator `LV_VECTOR_PATH_QUALITY_LOW`

enum `lv_vector_blend_t`

Values:

enumerator `LV_VECTOR_BLEND_SRC_OVER`

enumerator **LV_VECTOR_BLEND_SRC_IN**

enumerator **LV_VECTOR_BLEND_DST_OVER**

enumerator **LV_VECTOR_BLEND_DST_IN**

enumerator **LV_VECTOR_BLEND_SCREEN**

enumerator **LV_VECTOR_BLEND_MULTIPLY**

enumerator **LV_VECTOR_BLEND_NONE**

enumerator **LV_VECTOR_BLEND_ADDITIVE**

enumerator **LV_VECTOR_BLEND_SUBTRACTIVE**

enum **lv_vector_path_op_t**

Values:

enumerator **LV_VECTOR_PATH_OP_MOVE_TO**

enumerator **LV_VECTOR_PATH_OP_LINE_TO**

enumerator **LV_VECTOR_PATH_OP_QUAD_TO**

enumerator **LV_VECTOR_PATH_OP_CUBIC_TO**

enumerator **LV_VECTOR_PATH_OP_CLOSE**

enum **lv_vector_draw_style_t**

Values:

enumerator **LV_VECTOR_DRAW_STYLE_SOLID**

enumerator **LV_VECTOR_DRAW_STYLE_PATTERN**

enumerator **LV_VECTOR_DRAW_STYLE_GRADIENT**

enum **lv_vector_gradient_spread_t**

Values:

enumerator **LV_VECTOR_GRADIENT_SPREAD_PAD**

enumerator **LV_VECTOR_GRADIENT_SPREAD_REPEAT**

enumerator **LV_VECTOR_GRADIENT_SPREAD_REFLECT**

enum **lv_vector_gradient_style_t**

Values:

enumerator **LV_VECTOR_GRADIENT_STYLE_LINEAR**

enumerator **LV_VECTOR_GRADIENT_STYLE_RADIAL**

enum **lv_vector_fill_units_t**

Values:

enumerator **LV_VECTOR_FILL_UNITS_OBJECT_BOUNDING_BOX**

enumerator **LV_VECTOR_FILL_UNITS_USER_SPACE_ON_USE**

Relative coordinates relative to the object bounding box.

Functions

void **lv_matrix_transform_point**(const lv_matrix_t *matrix, lv_fpoint_t *point)

Transform the coordinates of a point using given matrix

Parametri

- **matrix** -- pointer to a matrix
- **point** -- pointer to a point

void **lv_matrix_transform_path**(const lv_matrix_t *matrix, lv_vector_path_t *path)

Transform all the coordinates of a path using given matrix

Parametri

- **matrix** -- pointer to a matrix
- **path** -- pointer to a path

lv_vector_path_t ***lv_vector_path_create**(*lv_vector_path_quality_t* quality)

Create a vector graphic path object

Parametri

quality -- the quality hint of path

Ritorna

pointer to the created path object

void **lv_vector_path_copy**(lv_vector_path_t *target_path, const lv_vector_path_t *path)

Copy a path data to another

Parametri

- **target_path** -- pointer to a path
- **path** -- pointer to source path

void **lv_vector_path_clear**(lv_vector_path_t *path)

Clear path data

Parametri

path -- pointer to a path

void **lv_vector_path_delete**(lv_vector_path_t *path)

Delete the graphic path object

Parametri

path -- pointer to a path

void **lv_vector_path_move_to**(lv_vector_path_t *path, const lv_fpoint_t *p)

Begin a new sub path and set a point to path

Parametri

- **path** -- pointer to a path

- **p** -- pointer to a `lv_fpoint_t` variable

void **lv_vector_path_line_to**(lv_vector_path_t *path, const lv_fpoint_t *p)

Add a line to the path from last point to the point

Parametri

- **path** -- pointer to a path
- **p** -- pointer to a `lv_fpoint_t` variable

void **lv_vector_path_quad_to**(lv_vector_path_t *path, const lv_fpoint_t *p1, const lv_fpoint_t *p2)

Add a quadratic bezier line to the path from last point to the point

Parametri

- **path** -- pointer to a path
- **p1** -- pointer to a `lv_fpoint_t` variable for control point
- **p2** -- pointer to a `lv_fpoint_t` variable for end point

void **lv_vector_path_cubic_to**(lv_vector_path_t *path, const lv_fpoint_t *p1, const lv_fpoint_t *p2, const lv_fpoint_t *p3)

Add a cubic bezier line to the path from last point to the point

Parametri

- **path** -- pointer to a path
- **p1** -- pointer to a `lv_fpoint_t` variable for first control point
- **p2** -- pointer to a `lv_fpoint_t` variable for second control point
- **p3** -- pointer to a `lv_fpoint_t` variable for end point

void **lv_vector_path_arc_to**(lv_vector_path_t *path, float radius_x, float radius_y, float rotate_angle, bool large_arc, bool clockwise, const lv_fpoint_t *p)

Add ellipse arc to the path from last point to the point

Parametri

- **path** -- pointer to a path
- **radius_x** -- the x radius for ellipse arc
- **radius_y** -- the y radius for ellipse arc
- **rotate_angle** -- the rotate angle for arc
- **large_arc** -- true for large arc, otherwise small
- **clockwise** -- true for clockwise, otherwise anticlockwise
- **p** -- pointer to a `lv_fpoint_t` variable for end point

void **lv_vector_path_close**(lv_vector_path_t *path)

Close the sub path

Parametri

path -- pointer to a path

void **lv_vector_path_get_bounding**(const lv_vector_path_t *path, *lv_area_t* *area)

Get the bounding box of a path

Parametri

- **path** -- pointer to a path
- **area** -- pointer to a *`lv_area_t`* variable for bounding box

void **lv_vector_path_append_rect**(lv_vector_path_t *path, const *lv_area_t* *rect, float rx, float ry)

Add a rectangle to the path

Parametri

- **path** -- pointer to a path
- **rect** -- pointer to a *`lv_area_t`* variable
- **rx** -- the horizontal radius for rounded rectangle
- **ry** -- the vertical radius for rounded rectangle

void **lv_vector_path_append_circle**(lv_vector_path_t *path, const lv_fpoint_t *c, float rx, float ry)

Add a circle to the path

Parametri

- **path** -- pointer to a path
- **c** -- pointer to a `lv_fpoint_t` variable for center of the circle

- **rx** -- the horizontal radius for circle
- **ry** -- the vertical radius for circle

void **lv_vector_path_append_arc**(lv_vector_path_t *path, const lv_fpoint_t *c, float radius, float start_angle, float sweep, bool pie)

Add a arc to the path

Parametri

- **path** -- pointer to a path
- **c** -- pointer to a `lv_fpoint_t` variable for center of the circle
- **radius** -- the radius for arc
- **start_angle** -- the start angle for arc
- **sweep** -- the sweep angle for arc, could be negative
- **pie** -- true: draw a pie, false: draw a arc

void **lv_vector_path_append_path**(lv_vector_path_t *path, const lv_vector_path_t *subpath)

Add an sub path to the path

Parametri

- **path** -- pointer to a path
- **subpath** -- pointer to another path which will be added

lv_vector_dsc_t ***lv_vector_dsc_create**(lv_layer_t *layer)

Create a vector graphic descriptor

Parametri

layer -- pointer to a layer

Ritorna

pointer to the created descriptor

void **lv_vector_dsc_delete**(lv_vector_dsc_t *dsc)

Delete the vector graphic descriptor

Parametri

dsc -- pointer to a vector graphic descriptor

void **lv_vector_dsc_set_transform**(lv_vector_dsc_t *dsc, const lv_matrix_t *matrix)

Set a matrix to current transformation matrix

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **matrix** -- pointer to a matrix

void **lv_vector_dsc_set_blend_mode**(lv_vector_dsc_t *dsc, *lv_vector_blend_t* blend)

Set blend mode for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **blend** -- the blend mode to be set in `lv_vector_blend_t`

void **lv_vector_dsc_set_fill_color32**(lv_vector_dsc_t *dsc, *lv_color32_t* color)

Set fill color for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **color** -- the color to be set in *lv_color32_t* format

void **lv_vector_dsc_set_fill_color**(lv_vector_dsc_t *dsc, *lv_color_t* color)

Set fill color for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **color** -- the color to be set in *lv_color_t* format

void **lv_vector_dsc_set_fill_opa**(lv_vector_dsc_t *dsc, lv_opa_t opa)

Set fill opacity for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor

- **opa** -- the opacity to be set in lv_opa_t format

void **lv_vector_dsc_set_fill_rule**(lv_vector_dsc_t *dsc, *lv_vector_fill_t* rule)

Set fill rule for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **rule** -- the fill rule to be set in lv_vector_fill_t format

void **lv_vector_dsc_set_fill_units**(lv_vector_dsc_t *dsc, const *lv_vector_fill_units_t* units)

Set the fill units for descriptor.

> **Nota**
>
> The units can be either relative to the object bounding box or absolute in user space. This API specifically affects the drawing position of the fill image and does not impact other elements.

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **units** -- the units to be set in *lv_vector_fill_units_t* format

void **lv_vector_dsc_set_fill_image**(lv_vector_dsc_t *dsc, const lv_draw_image_dsc_t *img_dsc)

Set fill image for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **img_dsc** -- pointer to a `lv_draw_image_dsc_t` variable

void **lv_vector_dsc_set_fill_linear_gradient**(lv_vector_dsc_t *dsc, float x1, float y1, float x2, float y2)

Set fill linear gradient for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **x1** -- the x for start point
- **y1** -- the y for start point
- **x2** -- the x for end point
- **y2** -- the y for end point

void **lv_vector_dsc_set_fill_radial_gradient**(lv_vector_dsc_t *dsc, float cx, float cy, float radius)

Set fill radial gradient radius for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **cx** -- the x for center of the circle
- **cy** -- the y for center of the circle
- **radius** -- the radius for circle

void **lv_vector_dsc_set_fill_gradient_spread**(lv_vector_dsc_t *dsc, *lv_vector_gradient_spread_t* spread)

Set fill radial gradient spread for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **spread** -- the gradient spread to be set in lv_vector_gradient_spread_t format

void **lv_vector_dsc_set_fill_gradient_color_stops**(lv_vector_dsc_t *dsc, const *lv_grad_stop_t* *stops, uint16_t count)

Set fill gradient color stops for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **stops** -- an array of `lv_grad_stop_t` variables

- **count** -- the number of stops in the array, range: 0..LV_GRADIENT_MAX_STOPS

void **lv_vector_dsc_set_fill_transform**(lv_vector_dsc_t *dsc, const lv_matrix_t *matrix)

Set a matrix to current fill transformation matrix

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **matrix** -- pointer to a matrix

void **lv_vector_dsc_set_stroke_color32**(lv_vector_dsc_t *dsc, *lv_color32_t* color)

Set stroke color for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **color** -- the color to be set in *lv_color32_t* format

void **lv_vector_dsc_set_stroke_color**(lv_vector_dsc_t *dsc, *lv_color_t* color)

Set stroke color for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **color** -- the color to be set in *lv_color_t* format

void **lv_vector_dsc_set_stroke_opa**(lv_vector_dsc_t *dsc, lv_opa_t opa)

Set stroke opacity for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **opa** -- the opacity to be set in lv_opa_t format

void **lv_vector_dsc_set_stroke_width**(lv_vector_dsc_t *dsc, float width)

Set stroke line width for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **width** -- the stroke line width

void **lv_vector_dsc_set_stroke_dash**(lv_vector_dsc_t *dsc, float *dash_pattern, uint16_t dash_count)

Set stroke line dash pattern for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **dash_pattern** -- an array of values that specify the segments of dash line
- **dash_count** -- the length of dash pattern array

void **lv_vector_dsc_set_stroke_cap**(lv_vector_dsc_t *dsc, *lv_vector_stroke_cap_t* cap)

Set stroke line cap style for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **cap** -- the line cap to be set in lv_vector_stroke_cap_t format

void **lv_vector_dsc_set_stroke_join**(lv_vector_dsc_t *dsc, *lv_vector_stroke_join_t* join)

Set stroke line join style for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **join** -- the line join to be set in lv_vector_stroke_join_t format

void **lv_vector_dsc_set_stroke_miter_limit**(lv_vector_dsc_t *dsc, uint16_t miter_limit)

Set stroke miter limit for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **miter_limit** -- the stroke miter_limit

void **lv_vector_dsc_set_stroke_linear_gradient**(lv_vector_dsc_t *dsc, float x1, float y1, float x2, float y2)

Set stroke linear gradient for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **x1** -- the x for start point
- **y1** -- the y for start point
- **x2** -- the x for end point
- **y2** -- the y for end point

void **lv_vector_dsc_set_stroke_radial_gradient**(lv_vector_dsc_t *dsc, float cx, float cy, float radius)

Set stroke radial gradient for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **cx** -- the x for center of the circle
- **cy** -- the y for center of the circle
- **radius** -- the radius for circle

void **lv_vector_dsc_set_stroke_gradient_spread**(lv_vector_dsc_t *dsc, *lv_vector_gradient_spread_t* spread)

Set stroke color stops for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **spread** -- the gradient spread to be set in lv_vector_gradient_spread_t format

void **lv_vector_dsc_set_stroke_gradient_color_stops**(lv_vector_dsc_t *dsc, const *lv_grad_stop_t* *stops, uint16_t count)

Set stroke color stops for descriptor

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **stops** -- an array of *lv_grad_stop_t* variables
- **count** -- the number of stops in the array

void **lv_vector_dsc_set_stroke_transform**(lv_vector_dsc_t *dsc, const lv_matrix_t *matrix)

Set a matrix to current stroke transformation matrix

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **matrix** -- pointer to a matrix

void **lv_vector_dsc_identity**(lv_vector_dsc_t *dsc)

Set current transformation matrix to identity matrix

Parametri

dsc -- pointer to a vector graphic descriptor

void **lv_vector_dsc_scale**(lv_vector_dsc_t *dsc, float scale_x, float scale_y)

Change the scale factor of current transformation matrix

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **scale_x** -- the scale factor for the X direction
- **scale_y** -- the scale factor for the Y direction

void **lv_vector_dsc_rotate**(lv_vector_dsc_t *dsc, float degree)

Rotate current transformation matrix with origin

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **degree** -- angle to rotate

void **lv_vector_dsc_translate**(lv_vector_dsc_t *dsc, float tx, float ty)

Translate current transformation matrix to new position

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **tx** -- the amount of translate in x direction

- **tx** -- the amount of translate in y direction

void **lv_vector_dsc_skew**(lv_vector_dsc_t *dsc, float skew_x, float skew_y)

Change the skew factor of current transformation matrix

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **skew_x** -- the skew factor for x direction
- **skew_y** -- the skew factor for y direction

void **lv_vector_dsc_add_path**(lv_vector_dsc_t *dsc, const lv_vector_path_t *path)

Add a graphic path to the draw list

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **path** -- pointer to a path

void **lv_vector_clear_area**(lv_vector_dsc_t *dsc, const *lv_area_t* *rect)

Clear a rectangle area use current fill color

Parametri

- **dsc** -- pointer to a vector graphic descriptor
- **rect** -- the area to clear in the buffer

void **lv_draw_vector**(lv_vector_dsc_t *dsc)

Draw all the vector graphic paths

Parametri

dsc -- pointer to a vector graphic descriptor

struct **_lv_fpoint_t**

Public Members

float **x**

float **y**

14.12.19 lv_draw_vector_private.h

Functions

void **lv_vector_for_each_destroy_tasks**(*lv_ll_t* *task_list, *vector_draw_task_cb* cb, void *data)

struct **_lv_vector_path_t**

Public Members

lv_vector_path_quality_t **quality**

lv_array_t **ops**

lv_array_t **points**

struct **_lv_vector_gradient_t**

Public Members

lv_vector_gradient_style_t **style**

lv_grad_stop_t **stops**[2]

A gradient stop array

uint16_t **stops_count**

The number of used stops in the array

float **x1**

float **y1**

float **x2**

float **y2**

float **cx**

float **cy**

float **cr**

lv_vector_gradient_spread_t **spread**

struct **_lv_vector_fill_dsc_t**

Public Members

lv_vector_draw_style_t **style**

lv_color32_t **color**

lv_opa_t **opa**

lv_vector_fill_t **fill_rule**

lv_vector_fill_units_t **fill_units**

lv_draw_image_dsc_t **img_dsc**

lv_vector_gradient_t **gradient**

lv_matrix_t **matrix**

struct **_lv_vector_stroke_dsc_t**

Public Members

lv_vector_draw_style_t **style**

lv_color32_t **color**

lv_opa_t **opa**

float **width**

lv_array_t **dash_pattern**

lv_vector_stroke_cap_t **cap**

lv_vector_stroke_join_t **join**

uint16_t **miter_limit**

lv_vector_gradient_t **gradient**

lv_matrix_t **matrix**

struct **_lv_vector_draw_dsc_t**

Public Members

lv_vector_fill_dsc_t **fill_dsc**

lv_vector_stroke_dsc_t **stroke_dsc**

lv_matrix_t **matrix**

lv_vector_blend_t **blend_mode**

lv_area_t **scissor_area**

struct **_lv_draw_vector_task_dsc_t**

Public Members

lv_draw_dsc_base_t **base**

lv_ll_t ***task_list**

struct **_lv_vector_dsc_t**

Public Members

lv_layer_t *`layer`

lv_vector_draw_dsc_t `current_dsc`

lv_draw_vector_task_dsc_t `tasks`

14.12.20 lv_image_decoder.h

Typedef

typedef lv_result_t (*`lv_image_decoder_info_f_t`)(lv_image_decoder_t *decoder, lv_image_decoder_dsc_t *dsc, *lv_image_header_t* *header)

Get info from an image and store in the `header`

Param decoder
pointer to decoder object

Param dsc
pointer to decoder descriptor

Param header
store the info here

Return
LV_RESULT_OK: info written correctly; LV_RESULT_INVALID: failed

typedef lv_result_t (*`lv_image_decoder_open_f_t`)(lv_image_decoder_t *decoder, lv_image_decoder_dsc_t *dsc)

Open an image for decoding. Prepare it as it is required to read it later

Param decoder
pointer to the decoder the function associated with

Param dsc
pointer to decoder descriptor. `src`, `color` are already initialized in it.

typedef lv_result_t (*`lv_image_decoder_get_area_cb_t`)(lv_image_decoder_t *decoder, lv_image_decoder_dsc_t *dsc, const *lv_area_t* *full_area, *lv_area_t* *decoded_area)

Decode `full_area` pixels incrementally by calling in a loop. Set `decoded_area` values to `LV_COORD_MIN` on first call. Required only if the "open" function can't return with the whole decoded pixel array.

Param decoder
pointer to the decoder the function associated with

Param dsc
pointer to decoder descriptor

Param full_area
input parameter. the full area to decode after enough subsequent calls

Param decoded_area
input+output parameter. set the values to `LV_COORD_MIN` for the first call and to reset decoding. the decoded area is stored here after each call.

Return
LV_RESULT_OK: ok; LV_RESULT_INVALID: failed or there is nothing left to decode

typedef void (*`lv_image_decoder_close_f_t`)(lv_image_decoder_t *decoder, lv_image_decoder_dsc_t *dsc)

Close the pending decoding. Free resources etc.

Param decoder
pointer to the decoder the function associated with

Param dsc
pointer to decoder descriptor

typedef void (*`lv_image_decoder_custom_draw_t`)(lv_layer_t *layer, const lv_image_decoder_dsc_t *dsc, const *lv_area_t* *coords, const lv_draw_image_dsc_t *draw_dsc, const *lv_area_t* *clip_area)

Custom drawing functions for special image formats.

Param layer
pointer to a layer

Param dsc
pointer to decoder descriptor

Param coords
the coordinates of the image

Param draw_dsc
the draw image descriptor

Param clip_area
the clip area of the image

Enums

enum `lv_image_src_t`

Source of image.

Values:

enumerator `LV_IMAGE_SRC_VARIABLE`

enumerator `LV_IMAGE_SRC_FILE`

Binary/C variable

enumerator `LV_IMAGE_SRC_SYMBOL`

File in filesystem

enumerator `LV_IMAGE_SRC_UNKNOWN`

Symbol (lv_symbol_def.h)

Functions

lv_result_t `lv_image_decoder_get_info`(const void *src, *lv_image_header_t* *header)

Get information about an image. Try the created image decoder one by one. Once one is able to get info that info will be used.

Parametri

- **`src`** -- the image source. Can be 1) File name: E.g. "S:folder/img1.png" (The drivers needs to registered via *`lv_fs_drv_register()`*) 2) Variable: Pointer to an *`lv_image_dsc_t`* variable 3) Symbol: E.g. `LV_SYMBOL_OK`
- **`header`** -- the image info will be stored here

Ritorna
LV_RESULT_OK: success; LV_RESULT_INVALID: wasn't able to get info about the image

lv_result_t `lv_image_decoder_open`(lv_image_decoder_dsc_t *dsc, const void *src, const lv_image_decoder_args_t *args)

Open an image. Try the created image decoders one by one. Once one is able to open the image that decoder is saved in `dsc`

Parametri

- **`dsc`** -- describes a decoding session. Simply a pointer to an `lv_image_decoder_dsc_t` variable.

- **src** -- the image source. Can be 1) File name: E.g. "S:folder/img1.png" (The drivers needs to registered via *lv_fs_drv_register()*) 2) Variable: Pointer to an *lv_image_dsc_t* variable 3) Symbol: E.g. LV_SYMBOL_OK
- **args** -- args about how the image should be opened.

Ritorna

LV_RESULT_OK: opened the image. dsc->decoded and dsc->header are set. LV_RESULT_INVALID: none of the registered image decoders were able to open the image.

lv_result_t **lv_image_decoder_get_area**(lv_image_decoder_dsc_t *dsc, const *lv_area_t* *full_area, *lv_area_t* *decoded_area)

Decode full_area pixels incrementally by calling in a loop. Set decoded_area to LV_COORD_MIN on first call.

Parametri

- **dsc** -- image decoder descriptor
- **full_area** -- input parameter. the full area to decode after enough subsequent calls
- **decoded_area** -- input+output parameter. set the values to LV_COORD_MIN for the first call and to reset decoding. the decoded area is stored here after each call.

Ritorna

LV_RESULT_OK: success; LV_RESULT_INVALID: an error occurred or there is nothing left to decode

void **lv_image_decoder_close**(lv_image_decoder_dsc_t *dsc)

Close a decoding session

Parametri

dsc -- pointer to lv_image_decoder_dsc_t used in lv_image_decoder_open

lv_image_decoder_t ***lv_image_decoder_create**(void)

Create a new image decoder

Ritorna

pointer to the new image decoder

void **lv_image_decoder_delete**(lv_image_decoder_t *decoder)

Delete an image decoder

Parametri

decoder -- pointer to an image decoder

lv_image_decoder_t ***lv_image_decoder_get_next**(lv_image_decoder_t *decoder)

Get the next image decoder in the linked list of image decoders

Parametri

decoder -- pointer to an image decoder or NULL to get the first one

Ritorna

the next image decoder or NULL if no more image decoder exists

void **lv_image_decoder_set_info_cb**(lv_image_decoder_t *decoder, *lv_image_decoder_info_f_t* info_cb)

Set a callback to get information about the image

Parametri

- **decoder** -- pointer to an image decoder
- **info_cb** -- a function to collect info about an image (fill an *lv_image_header_t* struct)

void **lv_image_decoder_set_open_cb**(lv_image_decoder_t *decoder, *lv_image_decoder_open_f_t* open_cb)

Set a callback to open an image

Parametri

- **decoder** -- pointer to an image decoder
- **open_cb** -- a function to open an image

void **lv_image_decoder_set_get_area_cb**(lv_image_decoder_t *decoder, *lv_image_decoder_get_area_cb_t* read_line_cb)

Set a callback to a decoded line of an image

Parametri

- **decoder** -- pointer to an image decoder
- **read_line_cb** -- a function to read a line of an image

void **lv_image_decoder_set_close_cb**(lv_image_decoder_t *decoder, *lv_image_decoder_close_f_t* close_cb)

Set a callback to close a decoding session. E.g. close files and free other resources.

Parametri

- **decoder** -- pointer to an image decoder
- **close_cb** -- a function to close a decoding session

lv_cache_entry_t ***lv_image_decoder_add_to_cache**(lv_image_decoder_t *decoder, lv_image_cache_data_t *search_key, const lv_draw_buf_t *decoded, void *user_data)

lv_draw_buf_t ***lv_image_decoder_post_process**(lv_image_decoder_dsc_t *dsc, lv_draw_buf_t *decoded)

Check the decoded image, make any modification if decoder `args` requires.

> **Nota**
>
> A new draw buf will be allocated if provided `decoded` is not modifiable or stride mismatch etc.

Parametri

- **dsc** -- pointer to a decoder descriptor
- **decoded** -- pointer to a decoded image to post process to meet dsc->args requirement.

Ritorna

post processed draw buffer, when it differs with `decoded`, it's newly allocated.

14.12.21 lv_image_decoder_private.h

Functions

void **lv_image_decoder_init**(uint32_t image_cache_size, uint32_t image_header_count)

Initialize the image decoder module

Parametri

- **image_cache_size** -- Image cache size in bytes. 0 to disable cache.
- **image_header_count** -- Number of header cache entries. 0 to disable header cache.

void **lv_image_decoder_deinit**(void)

Deinitialize the image decoder module

struct **_lv_image_decoder_args_t**

#include <lv_image_decoder_private.h> Image decoder args. It determines how to decoder an image, e.g. whether to premultiply the alpha or not. It should be passed to lv_img_decoder_open() function. If NULL is provided, default args are used.

Default args: all field are zero or false.

Public Members

bool **stride_align**

Whether stride should be aligned

bool **premultiply**

Whether image should be premultiplied or not after decoding

bool **no_cache**

When set, decoded image won't be put to cache, and decoder open will also ignore cache.

bool **use_indexed**

Decoded indexed image as is. Convert to ARGB8888 if false.

bool **flush_cache**

Whether to flush the data cache after decoding

struct **_lv_image_decoder_t**

Public Members

lv_image_decoder_info_f_t **info_cb**

lv_image_decoder_open_f_t **open_cb**

lv_image_decoder_get_area_cb_t **get_area_cb**

lv_image_decoder_close_f_t **close_cb**

lv_image_decoder_custom_draw_t **custom_draw_cb**

const char ***name**

void ***user_data**

struct **_lv_image_cache_data_t**

Public Members

lv_cache_slot_size_t **slot**

const void ***src**

lv_image_src_t **src_type**

const lv_draw_buf_t ***decoded**

const lv_image_decoder_t ***decoder**

void ***user_data**

struct **_lv_image_header_cache_data_t**

Public Members

const void ***src**

lv_image_src_t **src_type**

lv_image_header_t **header**

lv_image_decoder_t ***decoder**

struct **_lv_image_decoder_dsc_t**

#include <lv_image_decoder_private.h> Describe an image decoding session. Stores data about the decoding

Public Members

lv_image_decoder_t ***decoder**

The decoder which was able to open the image source

lv_image_decoder_args_t **args**

A copy of parameters of how this image is decoded

const void ***src**

The image source. A file path like "S:my_img.png" or pointer to an `lv_image_dsc_t` variable

lv_image_src_t **src_type**

Type of the source: file or variable. Can be set in `open` function if required

lv_fs_file_t **file**

lv_image_header_t **header**

Info about the opened image: color format, size, etc. MUST be set in `open` function

const lv_draw_buf_t ***decoded**

Pointer to a draw buffer where the image's data (pixels) are stored in a decoded, plain format. MUST be set in `open` or `get_area_cb`function

const *lv_color32_t* ***palette**

uint32_t **palette_size**

uint32_t **time_to_open**

How much time did it take to open the image. [ms] If not set `lv_image_cache` will measure and set the time to open

const char ***error_msg**

A text to display instead of the image when the image can't be opened. Can be set in `open` function or set NULL.

lv_cache_t ***cache**

lv_cache_entry_t ***cache_entry**

Point to cache entry information

void ***user_data**

Store any custom data here is required

14.12.22 lv_image_dsc.h

Defines

LV_IMAGE_HEADER_MAGIC

Magic number for lvgl image, 9 means lvgl version 9 It must be neither a valid ASCII character nor larger than 0x80. See `lv_image_src_get_type`.

LV_IMAGE_FLAGS_USER_MASK

Flags reserved for user, lvgl won't use these bits.

Typedef

typedef enum *_lvimage_flags_t* **lv_image_flags_t**

Enums

enum **_lvimage_flags_t**

Values:

enumerator **LV_IMAGE_FLAGS_PREMULTIPLIED**

For RGB map of the image data, mark if it's pre-multiplied with alpha. For indexed image, this bit indicated palette data is pre-multiplied with alpha.

enumerator **LV_IMAGE_FLAGS_COMPRESSED**

The image data is compressed, so decoder needs to decode image firstly. If this flag is set, the whole image will be decompressed upon decode, and `get_area_cb` won't be necessary.

enumerator **LV_IMAGE_FLAGS_ALLOCATED**

The image is allocated from heap, thus should be freed after use.

enumerator **LV_IMAGE_FLAGS_MODIFIABLE**

If the image data is malloced and can be processed in place. In image decoder post processing, this flag means we modify it in-place.

enumerator **LV_IMAGE_FLAGS_CUSTOM_DRAW**

The image has custom drawing methods.

enumerator **LV_IMAGE_FLAGS_USER1**

Flags reserved for user, lvgl won't use these bits.

enumerator **LV_IMAGE_FLAGS_USER2**

enumerator **LV_IMAGE_FLAGS_USER3**

enumerator **LV_IMAGE_FLAGS_USER4**

enumerator **LV_IMAGE_FLAGS_USER5**

enumerator **LV_IMAGE_FLAGS_USER6**

enumerator **LV_IMAGE_FLAGS_USER7**

enumerator **LV_IMAGE_FLAGS_USER8**

enum **lv_image_compress_t**

Values:

enumerator **LV_IMAGE_COMPRESS_NONE**

enumerator **LV_IMAGE_COMPRESS_RLE**

LVGL custom RLE compression

enumerator **LV_IMAGE_COMPRESS_LZ4**

struct **lv_image_header_t**

Public Members

uint32_t **magic**

Magic number. Must be LV_IMAGE_HEADER_MAGIC

uint32_t **cf**

Color format: See *lv_color_format_t*

uint32_t **flags**

Image flags, see `lv_image_flags_t`

uint32_t **w**

uint32_t **h**

uint32_t **stride**

Number of bytes in a row

uint32_t **reserved_2**

Reserved to be used later

struct **lv_yuv_plane_t**

Public Members

void ***buf**

uint32_t **stride**

Number of bytes in a row

union **lv_yuv_buf_t**

Public Members

lv_yuv_plane_t **yuv**

packed format

lv_yuv_plane_t **y**

lv_yuv_plane_t **u**

lv_yuv_plane_t **v**

struct *lv_yuv_buf_t* **planar**

planar format with 3 plane

lv_yuv_plane_t **uv**

struct *lv_yuv_buf_t* **semi_planar**

planar format with 2 plane

struct **lv_image_dsc_t**

#include <lv_image_dsc.h> Struct to describe a constant image resource. It's similar to lv_draw_buf_t, but the data is constant.

Public Members

lv_image_header_t **header**

A header describing the basics of the image

uint32_t **data_size**

Size of the image in bytes

const uint8_t ***data**

Pointer to the data of the image

const void ***reserved**

A reserved field to make it has same size as lv_draw_buf_t

const void ***reserved_2**

A reserved field to make it has same size as lv_draw_buf_t

14.12.23 dma2d

lv_draw_dma2d.h

Functions

void **lv_draw_dma2d_init**(void)

void **lv_draw_dma2d_deinit**(void)

void **lv_draw_dma2d_transfer_complete_interrupt_handler**(void)

lv_draw_dma2d_private.h

Defines

LV_DRAW_DMA2D_ASYNC

LV_DRAW_DMA2D_CACHE

Enums

enum **lv_draw_dma2d_output_cf_t**

Values:

enumerator **LV_DRAW_DMA2D_OUTPUT_CF_ARGB8888**

enumerator **LV_DRAW_DMA2D_OUTPUT_CF_RGB888**

enumerator **LV_DRAW_DMA2D_OUTPUT_CF_RGB565**

enumerator **LV_DRAW_DMA2D_OUTPUT_CF_ARGB1555**

enumerator **LV_DRAW_DMA2D_OUTPUT_CF_ARGB4444**

enum **lv_draw_dma2d_fgbg_cf_t**

Values:

enumerator **LV_DRAW_DMA2D_FGBG_CF_ARGB8888**

enumerator **LV_DRAW_DMA2D_FGBG_CF_RGB888**

enumerator **LV_DRAW_DMA2D_FGBG_CF_RGB565**

enumerator **LV_DRAW_DMA2D_FGBG_CF_ARGB1555**

enumerator **LV_DRAW_DMA2D_FGBG_CF_ARGB4444**

enumerator **LV_DRAW_DMA2D_FGBG_CF_L8**

enumerator **LV_DRAW_DMA2D_FGBG_CF_AL44**

enumerator **LV_DRAW_DMA2D_FGBG_CF_AL88**

enumerator **LV_DRAW_DMA2D_FGBG_CF_L4**

enumerator **LV_DRAW_DMA2D_FGBG_CF_A8**

enumerator **LV_DRAW_DMA2D_FGBG_CF_A4**

enumerator **LV_DRAW_DMA2D_FGBG_CF_YCBCR**

enum **lv_draw_dma2d_mode_t**

Values:

enumerator **LV_DRAW_DMA2D_MODE_MEMORY_TO_MEMORY**

enumerator **LV_DRAW_DMA2D_MODE_MEMORY_TO_MEMORY_WITH_PFC**

enumerator **LV_DRAW_DMA2D_MODE_MEMORY_TO_MEMORY_WITH_BLENDING**

enumerator **LV_DRAW_DMA2D_MODE_REGISTER_TO_MEMORY**

enumerator **LV_DRAW_DMA2D_MODE_MEMORY_TO_MEMORY_WITH_BLENDING_AND_FIXED_COLOR_FG**

enumerator **LV_DRAW_DMA2D_MODE_MEMORY_TO_MEMORY_WITH_BLENDING_AND_FIXED_COLOR_BG**

enum **lv_draw_dma2d_alpha_mode_t**

Values:

enumerator **LV_DRAW_DMA2D_ALPHA_MODE_NO_MODIFY_IMAGE_ALPHA_CHANNEL**

enumerator **LV_DRAW_DMA2D_ALPHA_MODE_REPLACE_ALPHA_CHANNEL**

enumerator **LV_DRAW_DMA2D_ALPHA_MODE_MULTIPLY_IMAGE_ALPHA_CHANNEL**

Functions

void **lv_draw_dma2d_opaque_fill**(lv_draw_task_t *t, void *first_pixel, int32_t w, int32_t h, int32_t stride)

void **lv_draw_dma2d_fill**(lv_draw_task_t *t, void *first_pixel, int32_t w, int32_t h, int32_t stride)

void **lv_draw_dma2d_opaque_image**(lv_draw_task_t *t, const lv_draw_image_dsc_t *draw_dsc, const *lv_area_t* *coords)

void **lv_draw_dma2d_image**(lv_draw_task_t *t, const lv_draw_image_dsc_t *draw_dsc, const *lv_area_t* *coords)

lv_draw_dma2d_output_cf_t **lv_draw_dma2d_cf_to_dma2d_output_cf**(*lv_color_format_t* cf)

uint32_t **lv_draw_dma2d_color_to_dma2d_color**(*lv_draw_dma2d_output_cf_t* cf, *lv_color_t* color)

void **lv_draw_dma2d_configure_and_start_transfer**(const *lv_draw_dma2d_configuration_t* *conf)

struct **lv_draw_dma2d_configuration_t**

Public Members

lv_draw_dma2d_mode_t **mode**

uint32_t **w**

uint32_t **h**

void ***output_address**

uint32_t **output_offset**

lv_draw_dma2d_output_cf_t **output_cf**

uint32_t **reg_to_mem_mode_color**

const void ***fg_address**

uint32_t **fg_offset**

lv_draw_dma2d_fgbg_cf_t **fg_cf**

uint32_t **fg_color**

uint32_t **fg_alpha_mode**

uint32_t **fg_alpha**

const void ***bg_address**

uint32_t **bg_offset**

lv_draw_dma2d_fgbg_cf_t **bg_cf**

uint32_t **bg_color**

uint32_t **bg_alpha_mode**

uint32_t **bg_alpha**

struct **lv_draw_dma2d_cache_area_t**

Public Members

const void ***first_byte**

uint32_t **width_bytes**

uint32_t **height**

uint32_t **stride**

struct **lv_draw_dma2d_unit_t**

Public Members

lv_draw_unit_t **base_unit**

lv_draw_task_t *volatile **task_act**

14.12.24 espressif

ppa

lv_draw_ppa.h

Functions

void **lv_draw_ppa_init**(void)

void **lv_draw_ppa_deinit**(void)

void **lv_draw_buf_ppa_init_handlers**(void)

void **lv_draw_ppa_fill**(lv_draw_task_t *t, const *lv_draw_fill_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_ppa_img**(lv_draw_task_t *t, const lv_draw_image_dsc_t *dsc, const *lv_area_t* *coords)

lv_draw_ppa_private.h

Defines

LV_PPA_NONBLOCKING_OPS

PPA_ALIGN_UP(x, align)

PPA_PTR_ALIGN_UP(p, align)

PPA_ALIGN_DOWN(x, align)

PPA_PTR_ALIGN_DOWN(p, align)

Typedef

typedef struct *lv_draw_ppa_unit* **lv_draw_ppa_unit_t**

Functions

static inline bool **ppa_src_cf_supported**(*lv_color_format_t* cf)

static inline bool **ppa_dest_cf_supported**(*lv_color_format_t* cf)

static inline ppa_fill_color_mode_t **lv_color_format_to_ppa_fill**(*lv_color_format_t* lv_fmt)

static inline ppa_blend_color_mode_t **lv_color_format_to_ppa_blend**(*lv_color_format_t* lv_fmt)

static inline ppa_srm_color_mode_t **lv_color_format_to_ppa_srm**(*lv_color_format_t* lv_fmt)

struct **lv_draw_ppa_unit**

Public Members

lv_draw_unit_t **base_unit**

lv_draw_task_t ***task_act**

ppa_client_handle_t **srm_client**

ppa_client_handle_t **fill_client**

ppa_client_handle_t **blend_client**

uint8_t ***buf**

14.12.25 eve

lv_draw_eve.h

Functions

void **lv_draw_eve_init**(void)

void **lv_draw_eve_set_display_data**(lv_display_t *disp, const *lv_draw_eve_parameters_t* *params, *lv_draw_eve_operation_cb_t* op_cb)

lv_draw_eve_private.h

Defines

DEGREES(n)

F16(x)

lv_draw_eve_unit_g

Functions

void **lv_draw_eve_image**(lv_draw_task_t *t, const lv_draw_image_dsc_t *draw_dsc, const *lv_area_t* *coords)

void **lv_draw_eve_fill**(lv_draw_task_t *t, const *lv_draw_fill_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_eve_border**(lv_draw_task_t *t, const *lv_draw_border_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_eve_line**(lv_draw_task_t *t, const *lv_draw_line_dsc_t* *dsc)

void **lv_draw_eve_label**(lv_draw_task_t *t, const *lv_draw_label_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_eve_arc**(lv_draw_task_t *t, const *lv_draw_arc_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_eve_triangle**(lv_draw_task_t *t, const *lv_draw_triangle_dsc_t* *dsc)

struct **lv_draw_eve_ramg_hash_table_cell_t**

Public Members

uintptr_t **key**

uint32_t **addr**

struct **lv_draw_eve_ramg_t**

Public Members

uint32_t **ramg_addr_end**

uint32_t **hash_table_cell_count**

uint32_t **hash_table_cells_occupied**

lv_draw_eve_ramg_hash_table_cell_t ***hash_table**

struct **_lv_draw_eve_unit_t**

Public Members

lv_draw_unit_t **base_unit**

lv_draw_task_t ***task_act**

lv_display_t ***disp**

lv_draw_eve_ramg_t **ramg**

lv_draw_eve_parameters_t **params**

lv_draw_eve_operation_cb_t **op_cb**

lv_draw_eve_ram_g.h

Defines

LV_DRAW_EVE_RAMG_OUT_OF_RAMG

Functions

bool **lv_draw_eve_ramg_get_addr**(uint32_t *addr_dst, uintptr_t key, uint32_t addr_size, uint32_t addr_align)

lv_draw_eve_target.h

Typedef

typedef void (***lv_draw_eve_operation_cb_t**)(lv_display_t *disp, *lv_draw_eve_operation_t* operation, void *data, uint32_t length)

Enums

enum **lv_draw_eve_operation_t**

Values:

enumerator **LV_DRAW_EVE_OPERATION_POWERDOWN_SET**

set the "PD_N" pin low

enumerator **LV_DRAW_EVE_OPERATION_POWERDOWN_CLEAR**

set the "PD_N" pin high

enumerator **LV_DRAW_EVE_OPERATION_CS_ASSERT**

set the "CS_N" pin low

enumerator **LV_DRAW_EVE_OPERATION_CS_DEASSERT**

set the "CS_N" pin high

enumerator **LV_DRAW_EVE_OPERATION_SPI_SEND**

send `length` bytes of `data` over SPI

enumerator **LV_DRAW_EVE_OPERATION_SPI_RECEIVE**

receive `length` bytes into `data` from SPI

struct **lv_draw_eve_parameters_t**

Public Members

uint16_t **hor_res**

active display width

uint16_t **ver_res**

active display height

uint16_t **hcycle**

total number of clocks per line, incl front/back porch

uint16_t **hoffset**

start of active line

uint16_t **hsync0**

start of horizontal sync pulse

uint16_t **hsync1**

end of horizontal sync pulse

uint16_t **vcycle**

total number of lines per screen, including pre/post

uint16_t **voffset**

start of active screen

uint16_t **vsync0**

start of vertical sync pulse

uint16_t **vsync1**

end of vertical sync pulse

uint8_t **swizzle**

FT8xx output to LCD - pin order

uint8_t **pclkpol**

LCD data is clocked in on this PCLK edge

uint8_t **cspread**

helps with noise, when set to 1 fewer signals are changed simultaneously, reset-default: 1

uint8_t **pclk**

60MHz / pclk = pclk frequency

bool **has_crystal**

has an external clock crystal

bool **has_gt911**

has a touch controller

uint8_t **backlight_pwm**

backlight PWM duty cycle 0 = off, 128 = max

uint16_t **backlight_freq**

backlight PWM frequency. try 4000 if unsure

lv_eve.h

Enums

enum **lv_eve_primitive_t**

Values:

enumerator **LV_EVE_PRIMITIVE_ZERO_VALUE**

enumerator **LV_EVE_PRIMITIVE_BITMAPS**

enumerator **LV_EVE_PRIMITIVE_POINTS**

enumerator **LV_EVE_PRIMITIVE_LINES**

enumerator **LV_EVE_PRIMITIVE_LINE_STRIP**

enumerator **LV_EVE_PRIMITIVE_EDGE_STRIP_R**

enumerator **LV_EVE_PRIMITIVE_EDGE_STRIP_L**

enumerator **LV_EVE_PRIMITIVE_EDGE_STRIP_A**

enumerator **LV_EVE_PRIMITIVE_EDGE_STRIP_B**

enumerator **LV_EVE_PRIMITIVE_RECTS**

Functions

void **lv_eve_save_context**(void)

void **lv_eve_restore_context**(void)

void **lv_eve_scissor**(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2)

void **lv_eve_primitive**(uint8_t context)

void **lv_eve_color**(*lv_color_t* color)

void **lv_eve_color_opa**(lv_opa_t opa)

void **lv_eve_line_width**(int32_t width)

void **lv_eve_point_size**(uint16_t radius)

void **lv_eve_vertex_2f**(int16_t x, int16_t y)

void **lv_eve_color_mask**(uint8_t r, uint8_t g, uint8_t b, uint8_t a)

void **lv_eve_stencil_func**(uint8_t func, uint8_t ref, uint8_t mask)

void **lv_eve_stencil_op**(uint8_t sfail, uint8_t spass)

void **lv_eve_blend_func**(uint8_t sfail, uint8_t spass)

void **lv_eve_draw_circle_simple**(int16_t coord_x1, int16_t coord_y1, uint16_t radius_t)

void **lv_eve_draw_rect_simple**(int16_t coord_x1, int16_t coord_y1, int16_t coord_x2, int16_t coord_y2, uint16_t radius)

void **lv_eve_mask_round**(int16_t coord_x1, int16_t coord_y1, int16_t coord_x2, int16_t coord_y2, int16_t radius)

struct **lv_eve_drawing_context_t**

Public Members

lv_eve_primitive_t **primitive**

lv_color_t **color**

lv_opa_t **opa**

int32_t **line_width**

uint16_t **point_size**

uint8_t **color_mask**[4]

uint8_t **stencil_func**[3]

uint8_t **stencil_op**[2]

uint8_t **blend_func**[2]

uint16_t **scx**

uint16_t **scy**

14.12.26 nema_gfx

lv_draw_nema_gfx.h

MIT License

Copyright (c) 2008-24 Think Silicon Single Member PC

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice (including the next paragraph) shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Functions

void **lv_draw_nema_gfx_init**(void)

void **lv_draw_nema_gfx_deinit**(void)

void **lv_draw_nema_gfx_fill**(lv_draw_task_t *t, const *lv_draw_fill_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_nema_gfx_triangle**(lv_draw_task_t *t, const *lv_draw_triangle_dsc_t* *dsc)

void **lv_draw_nema_gfx_img**(lv_draw_task_t *t, const lv_draw_image_dsc_t *dsc, const *lv_area_t* *coords)

void **lv_draw_nema_gfx_label**(lv_draw_task_t *t, const *lv_draw_label_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_nema_gfx_label_init**(lv_draw_unit_t *draw_unit)

void **lv_draw_nema_gfx_layer**(lv_draw_task_t *t, const lv_draw_image_dsc_t *draw_dsc, const *lv_area_t* *coords)

void **lv_draw_nema_gfx_line**(lv_draw_task_t *t, const *lv_draw_line_dsc_t* *dsc)

void **lv_draw_nema_gfx_border**(lv_draw_task_t *t, const *lv_draw_border_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_nema_gfx_arc**(lv_draw_task_t *t, const *lv_draw_arc_dsc_t* *dsc, const *lv_area_t* *coords)

struct **lv_draw_nema_gfx_unit_t**

Public Members

lv_draw_unit_t **base_unit**

lv_draw_task_t ***task_act**

uint32_t **idx**

nema_cmdlist_t **cl**

NEMA_VG_PAINT_HANDLE **paint**

NEMA_VG_GRAD_HANDLE **gradient**

NEMA_VG_PATH_HANDLE **path**

lv_draw_nema_gfx_utils.h

MIT License

Copyright (c) 2008-24 Think Silicon Single Member PC

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice (including the next paragraph) shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Defines

NEMA_VIRT2PHYS

LV_NEMA_GFX_COLOR_FORMAT

LV_NEMA_GFX_FORMAT_MULTIPLIER

Functions

bool **lv_nemagfx_is_cf_supported**(*lv_color_format_t* cf)

Check if `lv_color_format_t` is supported.

Parametri

cf -- The LVGL color format

Ritorna

True/false

uint32_t **lv_nemagfx_cf_to_nema**(*lv_color_format_t* cf)

Convert a `lv_color_format_t` to a Nema color format.

Parametri

cf -- The LVGL color format

Ritorna

The Nema color format

uint32_t **lv_nemagfx_blending_mode**(*lv_blend_mode_t* lv_blend_mode)

Get NemaGFX blending mode

Parametri

lv_blend_mode -- **[in]** The LVGL blend mode

Ritorna
NemaGFX blending mode

void **lv_nemagfx_grad_set**(NEMA_VG_GRAD_HANDLE gradient, *lv_grad_dsc_t* lv_grad, lv_opa_t opa)

Get NemaGFX blending mode

Parametri

- **gradient** -- **[in]** NemaGFX Gradient Buffer
- **lv_grad** -- **[in]** Gradient descriptor
- **opa** -- **[in]** Descriptor's opacity

lv_nema_gfx_path.h

MIT License

Copyright (c) 2008-24 Think Silicon Single Member PC

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice (including the next paragraph) shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Functions

lv_nema_gfx_path_t ***lv_nema_gfx_path_create**(void)

void **lv_nema_gfx_path_alloc**(*lv_nema_gfx_path_t* *nema_gfx_path)

void **lv_nema_gfx_path_destroy**(*lv_nema_gfx_path_t* *nema_gfx_path)

void **lv_nema_gfx_path_move_to**(*lv_nema_gfx_path_t* *nema_gfx_path, float x, float y)

void **lv_nema_gfx_path_line_to**(*lv_nema_gfx_path_t* *nema_gfx_path, float x, float y)

void **lv_nema_gfx_path_quad_to**(*lv_nema_gfx_path_t* *nema_gfx_path, float cx, float cy, float x, float y)

void **lv_nema_gfx_path_cubic_to**(*lv_nema_gfx_path_t* *nema_gfx_path, float cx1, float cy1, float cx2, float cy2, float x, float y)

void **lv_nema_gfx_path_end**(*lv_nema_gfx_path_t* *nema_gfx_path)

struct **lv_nema_gfx_path_t**

Public Members

NEMA_VG_PATH_HANDLE **path**

NEMA_VG_PAINT_HANDLE **paint**

float ***data**

uint8_t ***seg**

uint32_t **data_size**

uint32_t **seg_size**

14.12.27 nxp

g2d

lv_draw_g2d.h

Typedef

typedef struct *lv_draw_g2d_unit* **lv_draw_g2d_unit_t**

Copyright 2024 NXP

SPDX-License-Identifier: MIT

Functions

void **lv_draw_g2d_init**(void)

void **lv_draw_g2d_deinit**(void)

void **lv_draw_buf_g2d_init_handlers**(void)

void **lv_draw_g2d_fill**(lv_draw_task_t *t)

void **lv_draw_g2d_img**(lv_draw_task_t *t)

struct **lv_draw_g2d_unit**

#include <lv_draw_g2d.h> Copyright 2024 NXP

SPDX-License-Identifier: MIT

Public Members

lv_draw_unit_t **base_unit**

lv_draw_task_t ***task_act**

void ***g2d_handle**

lv_g2d_buf_map.h

Typedef

typedef struct *lv_map_item* **lv_map_item_t**

Copyright 2024 NXP

SPDX-License-Identifier: MIT

typedef struct *lv_buf_map* **lv_buf_map_t**

Functions

void **g2d_create_buf_map**(void)

void **g2d_free_buf_map**(void)

void **g2d_insert_buf_map**(void *key, struct g2d_buf *value)

struct g2d_buf ***g2d_search_buf_map**(void *key)

void **g2d_free_item**(void *key)

void **g2d_print_table**(void)

struct **lv_map_item**

#include <lv_g2d_buf_map.h> Copyright 2024 NXP

SPDX-License-Identifier: MIT

Public Members

void ***key**

struct g2d_buf ***value**

struct **lv_buf_map**

Public Members

lv_map_item_t ****items**

lv_array_t ****overflow_list**

int **size**

int **count**

lv_g2d_utils.h

Defines

G2D_ASSERT(expr)

Copyright 2024 NXP

SPDX-License-Identifier: MIT

G2D_ASSERT_MSG(expr, msg)

Typedef

typedef enum g2d_format **g2d_format_t**

Functions

g2d_format_t **g2d_get_buf_format**(*lv_color_format_t* cf)

uint32_t **g2d_rgba_to_u32**(*lv_color_t* color)

int32_t **g2d_get_buf_fd**(const lv_draw_buf_t *draw_buf)

pxp

lv_draw_pxp.h

Typedef

typedef struct *_lv_draw_pxp_unit_t* **lv_draw_pxp_unit_t**

Copyright 2022-2024 NXP

SPDX-License-Identifier: MIT

Functions

void **lv_draw_pxp_init**(void)

void **lv_draw_pxp_deinit**(void)

void **lv_draw_pxp_rotate**(const void *src_buf, void *dest_buf, int32_t src_width, int32_t src_height, int32_t src_stride, int32_t dest_stride, *lv_display_rotation_t* rotation, *lv_color_format_t* cf)

void **lv_draw_buf_pxp_init_handlers**(void)

void **lv_draw_pxp_fill**(lv_draw_task_t *t)

void **lv_draw_pxp_img**(lv_draw_task_t *t)

void **lv_draw_pxp_layer**(lv_draw_task_t *t)

struct **_lv_draw_pxp_unit_t**

#include <lv_draw_pxp.h> Copyright 2022-2024 NXP

SPDX-License-Identifier: MIT

Public Members

lv_draw_unit_t **base_unit**

lv_draw_task_t ***task_act**

lv_pxp_cfg.h

Defines

PXP_ID

Copyright 2020-2023 NXP

SPDX-License-Identifier: MIT PXP module instance to use

PXP_IRQ_ID

PXP interrupt line ID

Functions

void **lv_pxp_init**(void)

Reset and initialize PXP device. This function should be called as a part of display init sequence.

void **lv_pxp_deinit**(void)

Disable PXP device. Should be called during display deinit sequence.

void **lv_pxp_reset**(void)

Reset PXP device.

void **lv_pxp_run**(void)

Clear cache and start PXP.

void **lv_pxp_wait**(void)

Wait for PXP completion.

struct **pxp_cfg_t**

#include <lv_pxp_cfg.h> NXP PXP device configuration.

Public Members

void (*__pxp_interrupt_init__)(void)

Callback for PXP interrupt initialization

void (*__pxp_interrupt_deinit__)(void)

Callback for PXP interrupt de-initialization

void (*__pxp_run__)(void)

Callback for PXP start

void (*__pxp_wait__)(void)

Callback for waiting of PXP completion

lv_pxp_osa.h

Functions

void **PXP_IRQHandler**(void)

Copyright 2020, 2022-2023 NXP

SPDX-License-Identifier: MIT PXP device interrupt handler. Used to check PXP task completion status.

pxp_cfg_t ***pxp_get_default_cfg**(void)

Get the PXP default configuration.

lv_pxp_utils.h

Defines

PXP_ASSERT(expr)

Copyright 2023-2024 NXP

SPDX-License-Identifier: MIT

PXP_ASSERT_MSG(expr, msg)

Functions

pxp_output_pixel_format_t **pxp_get_out_px_format**(*lv_color_format_t* cf)

pxp_as_pixel_format_t **pxp_get_as_px_format**(*lv_color_format_t* cf)

pxp_ps_pixel_format_t **pxp_get_ps_px_format**(*lv_color_format_t* cf)

bool **pxp_buf_aligned**(const void *buf, uint32_t stride)

vglite

lv_draw_vglite.h

Typedef

typedef struct *vglite_draw_task* **vglite_draw_task_t**

Copyright 2023-2024 NXP

SPDX-License-Identifier: MIT

typedef struct *lv_draw_vglite_unit* **lv_draw_vglite_unit_t**

Functions

void **lv_draw_buf_vglite_init_handlers**(void)

void **lv_draw_vglite_init**(void)

void **lv_draw_vglite_deinit**(void)

void **lv_draw_vglite_arc**(*vglite_draw_task_t* *vglite_task)

void **lv_draw_vglite_border**(*vglite_draw_task_t* *vglite_task)

void **lv_draw_vglite_fill**(*vglite_draw_task_t* *vglite_task)

void **lv_draw_vglite_img**(*vglite_draw_task_t* *vglite_task)

void **lv_draw_vglite_label**(*vglite_draw_task_t* *vglite_task)

void **lv_draw_vglite_layer**(*vglite_draw_task_t* *vglite_task)

void **lv_draw_vglite_line**(*vglite_draw_task_t* *vglite_task)

void **lv_draw_vglite_triangle**(*vglite_draw_task_t* *vglite_task)

struct **vglite_draw_task**

#include <lv_draw_vglite.h> Copyright 2023-2024 NXP

SPDX-License-Identifier: MIT

Public Members

lv_draw_task_t ***t**

vg_lite_path_t ***path**

vg_lite_linear_gradient_t ***gradient**

int32_t ***path_data**

struct **lv_draw_vglite_unit**

Public Members

lv_draw_unit_t **base_unit**

vglite_draw_task_t ***task_act**

lv_vglite_buf.h

Functions

vg_lite_buffer_t ***vglite_get_dest_buf**(void)

Copyright 2023 NXP

SPDX-License-Identifier: MIT Get vglite destination buffer pointer.

Valori restituiti

The -- vglite destination buffer

vg_lite_buffer_t ***vglite_get_src_buf**(void)

Get vglite source buffer pointer.

Valori restituiti

The -- vglite source buffer

void **vglite_set_dest_buf_ptr**(void *buf)

Set vglite destination buffer address only.

Parametri

buf -- **[in]** Destination buffer address (does not require alignment for VG_LITE_LINEAR mode)

void **vglite_set_src_buf_ptr**(const void *buf)

Set vglite source buffer address only.

Parametri

buf -- **[in]** Source buffer address

void **vglite_set_dest_buf**(const void *buf, uint32_t width, uint32_t height, uint32_t stride, *lv_color_format_t* cf)

Set vglite destination buffer.

Parametri

- **buf** -- **[in]** Destination buffer address
- **width** -- **[in]** Destination buffer width
- **height** -- **[in]** Destination buffer height
- **stride** -- **[in]** Destination buffer stride in bytes
- **cf** -- **[in]** Destination buffer color format

void **vglite_set_src_buf**(const void *buf, uint32_t width, uint32_t height, uint32_t stride, *lv_color_format_t* cf)

Set vglite source buffer.

Parametri

- **buf** -- **[in]** Source buffer address
- **width** -- **[in]** Source buffer width
- **height** -- **[in]** Source buffer height
- **stride** -- **[in]** Source buffer stride in bytes
- **cf** -- **[in]** Source buffer color format

void **vglite_set_buf**(vg_lite_buffer_t *vgbuf, void *buf, uint32_t width, uint32_t height, uint32_t stride, *lv_color_format_t* cf)

Set vglite buffer.

Parametri

- **vgbuf** -- **[in]** Address of the VGLite buffer object
- **buf** -- **[in]** Address of the memory for the VGLite buffer
- **width** -- **[in]** Buffer width
- **height** -- **[in]** Buffer height
- **stride** -- **[in]** Buffer stride in bytes
- **cf** -- **[in]** Buffer color format

lv_vglite_matrix.h

Functions

vg_lite_matrix_t ***vglite_get_matrix**(void)

Copyright 2023 NXP

SPDX-License-Identifier: MIT

void **vglite_set_translation_matrix**(const *lv_area_t* *dest_area)

Creates matrix that translates to origin of given destination area.

Parametri

dest_area -- **[in]** Area with relative coordinates of destination buffer

void **vglite_set_transformation_matrix**(const *lv_area_t* *dest_area, const lv_draw_image_dsc_t *dsc)

Creates matrix that translates to origin of given destination area with transformation (scale or rotate).

Parametri

- **dest_area** -- **[in]** Area with relative coordinates of destination buffer
- **dsc** -- **[in]** Image descriptor

lv_vglite_path.h

Defines

BEZIER_OPTIM_CIRCLE

Copyright 2023 NXP

SPDX-License-Identifier: MIT

BEZIER_DBG_CONTROL_POINTS

CUBIC_PATH_DATA_SIZE

LINE_PATH_DATA_SIZE

MOVE_PATH_DATA_SIZE

END_PATH_DATA_SIZE

RECT_PATH_DATA_MAX_SIZE

ARC_PATH_DATA_MAX_SIZE

Functions

void **vglite_create_rect_path_data**(int32_t *path_data, uint32_t *path_data_size, int32_t radius, const *lv_area_t* *coords)

Generates path data for rectangle drawing.

Parametri

- **[in/out]** -- path The path data to initialize
- **[in/out]** -- path_size The resulting size of the created path data
- **dsc** -- **[in]** The style descriptor for the rectangle to be drawn
- **coords** -- **[in]** The coordinates of the rectangle to be drawn

lv_vglite_utils.h

Defines

ENUM_TO_STRING(e)

Copyright 2022-2024 NXP

SPDX-License-Identifier: MIT

VGLITE_ASSERT(expr)

VGLITE_ASSERT_MSG(expr, msg)

VGLITE_CHECK_ERROR(function)

Functions

static inline void **vglite_set_scissor**(const *lv_area_t* *clip_area)

Set the clipping box.

Parametri

clip_area -- **[in]** Clip area with relative coordinates of destination buffer

const char ***vglite_error_to_string**(vg_lite_error_t error)

void **vglite_run**(void)

Flush command to VG-Lite.

vg_lite_color_t **vglite_get_color**(*lv_color32_t* lv_col32, bool gradient)

Wait for VG-Lite finish. Get vglite color. Premultiplies (if not hw already) and swizzles the given LVGL 32bit color to obtain vglite color.

Parametri

- **lv_col32** -- **[in]** The initial LVGL 32bit color
- **gradient** -- **[in]** True for gradient color

Valori restituiti

The -- vglite 32-bit color value:

vg_lite_blend_t **vglite_get_blend_mode**(*lv_blend_mode_t* lv_blend_mode)

Get vglite blend mode.

Parametri

lv_blend_mode -- **[in]** The LVGL blend mode

Valori restituiti

The -- vglite blend mode

vg_lite_buffer_format_t **vglite_get_buf_format**(*lv_color_format_t* cf)

Get vglite buffer format.

Parametri

cf -- **[in]** Color format

Valori restituiti

The -- vglite buffer format

uint8_t **vglite_get_stride_alignment**(*lv_color_format_t* cf)

Get vglite stride alignment.

Parametri

cf -- **[in]** Color format

Valori restituiti

Alignment -- requirement in bytes

bool **vglite_src_buf_aligned**(const void *buf, uint32_t stride, *lv_color_format_t* cf)

Check source start address and stride alignment.

Parametri

- **buf** -- **[in]** Buffer address
- **stride** -- **[in]** Stride of buffer in bytes
- **cf** -- **[in]** Color format - to calculate the expected alignment

Valori restituiti

true -- Alignment OK

14.12.28 opengles

lv_draw_opengles.h

Functions

void **lv_draw_opengles_init**(void)

void **lv_draw_opengles_deinit**(void)

14.12.29 renesas

dave2d

lv_draw_dave2d.h

Defines

D2_RENDER_EACH_OPERATION

Functions

void **lv_draw_dave2d_init**(void)

void **lv_draw_dave2d_image**(lv_draw_task_t *t, const lv_draw_image_dsc_t *draw_dsc, const *lv_area_t* *coords)

void **lv_draw_dave2d_fill**(lv_draw_task_t *t, const *lv_draw_fill_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_dave2d_border**(lv_draw_task_t *t, const *lv_draw_border_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_dave2d_box_shadow**(lv_draw_task_t *t, const *lv_draw_box_shadow_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_dave2d_label**(lv_draw_task_t *t, const *lv_draw_label_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_dave2d_arc**(lv_draw_task_t *t, const *lv_draw_arc_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_dave2d_line**(lv_draw_task_t *t, const *lv_draw_line_dsc_t* *dsc)

void **lv_draw_dave2d_layer**(lv_draw_task_t *t, const lv_draw_image_dsc_t *draw_dsc, const *lv_area_t* *coords)

void **lv_draw_dave2d_triangle**(lv_draw_task_t *t, const *lv_draw_triangle_dsc_t* *dsc)

void **lv_draw_dave2d_mask_rect**(lv_draw_task_t *t, const lv_draw_mask_rect_dsc_t *dsc, const *lv_area_t* *coords)

void **lv_draw_dave2d_transform**(lv_draw_task_t *t, const *lv_area_t* *dest_area, const void *src_buf, int32_t src_w, int32_t src_h, int32_t src_stride, const lv_draw_image_dsc_t *draw_dsc, const lv_draw_image_sup_t *sup, *lv_color_format_t* cf, void *dest_buf)

struct **lv_draw_dave2d_unit_t**

Public Members

lv_draw_unit_t **base_unit**

lv_draw_task_t ***task_act**

uint32_t **idx**

d2_device ***d2_handle**

d2_renderbuffer ***renderbuffer**

lv_draw_dave2d_utils.h

Functions

d2_color **lv_draw_dave2d_lv_colour_to_d2_colour**(*lv_color_t* color)

d2_s32 **lv_draw_dave2d_cf_fb_get**(void)

d2_u32 **lv_draw_dave2d_lv_colour_fmt_to_d2_fmt**(*lv_color_format_t* colour_format)

void **d2_framebuffer_from_layer**(d2_device *handle, lv_layer_t *layer)

bool **lv_draw_dave2d_is_dest_cf_supported**(*lv_color_format_t* cf)

14.12.30 sdl

lv_draw_sdl.h

Functions

void **lv_draw_sdl_init**(void)

void **lv_draw_sdl_image**(lv_draw_unit_t *draw_unit, const lv_draw_image_dsc_t *draw_dsc, const *lv_area_t* *coords)

void **lv_draw_sdl_fill**(lv_draw_unit_t *draw_unit, const *lv_draw_fill_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_sdl_border**(lv_draw_unit_t *draw_unit, const *lv_draw_border_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_sdl_box_shadow**(lv_draw_unit_t *draw_unit, const *lv_draw_box_shadow_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_sdl_label**(lv_draw_unit_t *draw_unit, const *lv_draw_label_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_sdl_arc**(lv_draw_unit_t *draw_unit, const *lv_draw_arc_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_sdl_line**(lv_draw_unit_t *draw_unit, const *lv_draw_line_dsc_t* *dsc)

void **lv_draw_sdl_layer**(lv_draw_unit_t *draw_unit, const lv_draw_image_dsc_t *draw_dsc, const *lv_area_t* *coords)

void **lv_draw_sdl_triangle**(lv_draw_unit_t *draw_unit, const *lv_draw_triangle_dsc_t* *dsc)

void **lv_draw_sdl_mask_rect**(lv_draw_unit_t *draw_unit, const lv_draw_mask_rect_dsc_t *dsc, const *lv_area_t* *coords)

14.12.31 sw

lv_draw_sw.h

Functions

void **lv_draw_sw_init**(void)

Initialize the SW renderer. Called in internally. It creates as many SW renderers as defined in LV_DRAW_SW_DRAW_UNIT_CNT

void **lv_draw_sw_deinit**(void)

Deinitialize the SW renderers

void **lv_draw_sw_fill**(lv_draw_task_t *t, *lv_draw_fill_dsc_t* *dsc, const *lv_area_t* *coords)

Fill an area using SW render. Handle gradient and radius.

Parametri

- **t** -- pointer to a draw task
- **dsc** -- the draw descriptor
- **coords** -- the coordinates of the rectangle

void **lv_draw_sw_border**(lv_draw_task_t *t, const *lv_draw_border_dsc_t* *dsc, const *lv_area_t* *coords)

Draw border with SW render.

Parametri

- **t** -- pointer to a draw task
- **dsc** -- the draw descriptor
- **coords** -- the coordinates of the rectangle

void **lv_draw_sw_box_shadow**(lv_draw_task_t *t, const *lv_draw_box_shadow_dsc_t* *dsc, const *lv_area_t* *coords)

Draw box shadow with SW render.

Parametri

- **t** -- pointer to a draw task
- **dsc** -- the draw descriptor
- **coords** -- the coordinates of the rectangle for which the box shadow should be drawn

void **lv_draw_sw_image**(lv_draw_task_t *t, const lv_draw_image_dsc_t *draw_dsc, const *lv_area_t* *coords)

Draw an image with SW render. It handles image decoding, tiling, transformations, and recoloring.

Parametri

- **t** -- pointer to a draw task
- **draw_dsc** -- the draw descriptor
- **coords** -- the coordinates of the image

void **lv_draw_sw_letter**(lv_draw_task_t *t, const *lv_draw_letter_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_sw_label**(lv_draw_task_t *t, const *lv_draw_label_dsc_t* *dsc, const *lv_area_t* *coords)

Draw a label with SW render.

Parametri

- **t** -- pointer to a draw task
- **dsc** -- the draw descriptor
- **coords** -- the coordinates of the label

void **lv_draw_sw_arc**(lv_draw_task_t *t, const *lv_draw_arc_dsc_t* *dsc, const *lv_area_t* *coords)

Draw an arc with SW render.

Parametri

- **t** -- pointer to a draw task
- **dsc** -- the draw descriptor
- **coords** -- the coordinates of the arc

void **lv_draw_sw_line**(lv_draw_task_t *t, const *lv_draw_line_dsc_t* *dsc)

Draw a line with SW render.

Parametri

- **t** -- pointer to a draw task
- **dsc** -- the draw descriptor

void **lv_draw_sw_layer**(lv_draw_task_t *t, const lv_draw_image_dsc_t *draw_dsc, const *lv_area_t* *coords)

Blend a layer with SW render

Parametri

- **t** -- pointer to a draw task
- **draw_dsc** -- the draw descriptor
- **coords** -- the coordinates of the layer

void **lv_draw_sw_triangle**(lv_draw_task_t *t, const *lv_draw_triangle_dsc_t* *dsc)

Draw a triangle with SW render.

Parametri

- **t** -- pointer to a draw task
- **dsc** -- the draw descriptor

void **lv_draw_sw_mask_rect**(lv_draw_task_t *t, const lv_draw_mask_rect_dsc_t *dsc)

Mask out a rectangle with radius from a current layer

Parametri

- **t** -- pointer to a draw task
- **dsc** -- the draw descriptor
- **coords** -- the coordinates of the mask

void **lv_draw_sw_transform**(const *lv_area_t* *dest_area, const void *src_buf, int32_t src_w, int32_t src_h, int32_t src_stride, const lv_draw_image_dsc_t *draw_dsc, const lv_draw_image_sup_t *sup, *lv_color_format_t* cf, void *dest_buf)

Used internally to get a transformed are of an image

Parametri

- **dest_area** -- area to calculate, i.e. get this area from the transformed image
- **src_buf** -- source buffer
- **src_w** -- source buffer width in pixels
- **src_h** -- source buffer height in pixels
- **src_stride** -- source buffer stride in bytes
- **draw_dsc** -- draw descriptor
- **sup** -- supplementary data
- **cf** -- color format of the source buffer
- **dest_buf** -- the destination buffer

void **lv_draw_sw_vector**(lv_draw_task_t *t, lv_draw_vector_task_dsc_t *dsc)

Draw vector graphics with SW render.

Parametri

- **t** -- pointer to a draw task

- **dsc** -- the draw descriptor

bool **lv_draw_sw_register_blend_handler**(*lv_draw_sw_custom_blend_handler_t* *handler)

Register a custom blend handler for a color format. Handler will be called when blending a color or an image to a buffer with the given color format. At most one handler can be registered for a color format. Subsequent registrations will overwrite the previous handler.

Parametri

handler -- pointer to a blend handler

Ritorna

true if the handler was registered, false if the handler could not be registered

bool **lv_draw_sw_unregister_blend_handler**(*lv_color_format_t* dest_cf)

Unregister a custom blend handler for a color format.

Parametri

dest_cf -- color format

Ritorna

true if a handler was unregistered, false if no handler was registered

lv_draw_sw_blend_handler_t **lv_draw_sw_get_blend_handler**(*lv_color_format_t* dest_cf)

Get the blend handler for a color format.

Parametri

dest_cf -- color format

Ritorna

pointer to the blend handler or NULL if no handler is registered

lv_draw_sw_grad.h

Functions

void **lv_draw_sw_grad_color_calculate**(const *lv_grad_dsc_t* *dsc, int32_t range, int32_t frac, *lv_color_t* *color_out, lv_opa_t *opa_out)

Compute the color in the given gradient and fraction Gradient are specified in a virtual [0-255] range, so this function scales the virtual range to the given range

Parametri

- **dsc** -- The gradient descriptor to use
- **range** -- The range to use in computation.
- **frac** -- The current part used in the range. frac is in [0; range]
- **color_out** -- Calculated gradient color
- **opa_out** -- Calculated opacity

lv_draw_sw_grad_calc_t ***lv_draw_sw_grad_get**(const *lv_grad_dsc_t* *gradient, int32_t w, int32_t h)

Get a gradient cache from the given parameters

void **lv_draw_sw_grad_cleanup**(*lv_draw_sw_grad_calc_t* *grad)

Clean up the gradient item after it was get with `lv_grad_get_from_cache`.

Parametri

grad -- pointer to a gradient

void **lv_draw_sw_grad_linear_setup**(*lv_grad_dsc_t* *dsc, const *lv_area_t* *coords)

Calculate constants from the given parameters that are used during rendering

Parametri

- **dsc** -- gradient descriptor
- **coords** -- the area where to draw the gradient

void **lv_draw_sw_grad_linear_cleanup**(*lv_grad_dsc_t* *dsc)

Free up the allocated memory for the gradient calculation

Parametri

dsc -- gradient descriptor

void **lv_draw_sw_grad_linear_get_line**(*lv_grad_dsc_t* *dsc, int32_t xp, int32_t yp, int32_t width, *lv_draw_sw_grad_calc_t* *result)

Calculate a line segment of a linear gradient

Parametri

- **dsc** -- gradient descriptor
- **xp** -- starting point x coordinate in gradient space
- **yp** -- starting point y coordinate in gradient space
- **width** -- width of the line segment in pixels
- **result** -- color buffer for the resulting line segment

void **lv_draw_sw_grad_radial_setup**(*lv_grad_dsc_t* *dsc, const *lv_area_t* *coords)

Calculate constants from the given parameters that are used during rendering

Parametri

- **dsc** -- gradient descriptor
- **coords** -- the area where to draw the gradient

void **lv_draw_sw_grad_radial_cleanup**(*lv_grad_dsc_t* *dsc)

Free up the allocated memory for the gradient calculation

Parametri

dsc -- gradient descriptor

void **lv_draw_sw_grad_radial_get_line**(*lv_grad_dsc_t* *dsc, int32_t xp, int32_t yp, int32_t width, *lv_draw_sw_grad_calc_t* *result)

Calculate a line segment of a radial gradient

Parametri

- **dsc** -- gradient descriptor
- **xp** -- starting point x coordinate in gradient space
- **yp** -- starting point y coordinate in gradient space
- **width** -- width of the line segment in pixels
- **result** -- color buffer for the resulting line segment

void **lv_draw_sw_grad_conical_setup**(*lv_grad_dsc_t* *dsc, const *lv_area_t* *coords)

Calculate constants from the given parameters that are used during rendering

Parametri

- **dsc** -- gradient descriptor
- **coords** -- the area where to draw the gradient

void **lv_draw_sw_grad_conical_cleanup**(*lv_grad_dsc_t* *dsc)

Free up the allocated memory for the gradient calculation

Parametri

dsc -- gradient descriptor

void **lv_draw_sw_grad_conical_get_line**(*lv_grad_dsc_t* *dsc, int32_t xp, int32_t yp, int32_t width, *lv_draw_sw_grad_calc_t* *result)

Calculate a line segment of a conical gradient

Parametri

- **dsc** -- gradient descriptor
- **xp** -- starting point x coordinate in gradient space
- **yp** -- starting point y coordinate in gradient space
- **width** -- width of the line segment in pixels
- **result** -- color buffer for the resulting line segment

struct **lv_draw_sw_grad_calc_t**

Public Members

lv_color_t ***color_map**

lv_opa_t ***opa_map**

uint32_t **size**

lv_draw_sw_mask.h

Defines

LV_MASK_ID_INV

LV_MASK_MAX_NUM

Typedef

typedef *lv_draw_sw_mask_res_t* (***lv_draw_sw_mask_xcb_t**)(lv_opa_t *mask_buf, int32_t abs_x, int32_t abs_y, int32_t len, void *p)

A common callback type for every mask type. Used internally by the library.

Enums

enum **lv_draw_sw_mask_res_t**

Values:

enumerator **LV_DRAW_SW_MASK_RES_TRANSP**

enumerator **LV_DRAW_SW_MASK_RES_FULL_COVER**

enumerator **LV_DRAW_SW_MASK_RES_CHANGED**

enumerator **LV_DRAW_SW_MASK_RES_UNKNOWN**

enum **lv_draw_sw_mask_type_t**

Values:

enumerator **LV_DRAW_SW_MASK_TYPE_LINE**

enumerator **LV_DRAW_SW_MASK_TYPE_ANGLE**

enumerator **LV_DRAW_SW_MASK_TYPE_RADIUS**

enumerator **LV_DRAW_SW_MASK_TYPE_FADE**

enumerator **LV_DRAW_SW_MASK_TYPE_MAP**

enum **lv_draw_sw_mask_line_side_t**

Values:

enumerator **LV_DRAW_SW_MASK_LINE_SIDE_LEFT**

enumerator **LV_DRAW_SW_MASK_LINE_SIDE_RIGHT**

enumerator **LV_DRAW_SW_MASK_LINE_SIDE_TOP**

enumerator **LV_DRAW_SW_MASK_LINE_SIDE_BOTTOM**

Functions

void **lv_draw_sw_mask_init**(void)

void **lv_draw_sw_mask_deinit**(void)

lv_draw_sw_mask_res_t **lv_draw_sw_mask_apply**(void *masks[], lv_opa_t *mask_buf, int32_t abs_x, int32_t abs_y, int32_t len)

Apply the added buffers on a line. Used internally by the library's drawing routines.

Parametri

- **masks** -- the masks list to apply, must be ended with NULL pointer in array.
- **mask_buf** -- store the result mask here. Has to be `len` byte long. Should be initialized with `0xFF`.
- **abs_x** -- absolute X coordinate where the line to calculate start
- **abs_y** -- absolute Y coordinate where the line to calculate start
- **len** -- length of the line to calculate (in pixel count)

Ritorna

One of these values:

- `LV_DRAW_MASK_RES_FULL_TRANSP`: the whole line is transparent. `mask_buf` is not set to zero
- `LV_DRAW_MASK_RES_FULL_COVER`: the whole line is fully visible. `mask_buf` is unchanged
- `LV_DRAW_MASK_RES_CHANGED`: `mask_buf` has changed, it shows the desired opacity of each pixel in the given line

void **lv_draw_sw_mask_free_param**(void *p)

Free the data from the parameter. It's called inside `lv_draw_sw_mask_remove_id` and `lv_draw_sw_mask_remove_custom` Needs to be called only in special cases when the mask is not added by `lv_draw_mask_add` and not removed by `lv_draw_mask_remove_id` or `lv_draw_mask_remove_custom`

Parametri

p -- pointer to a mask parameter

void **lv_draw_sw_mask_line_points_init**(lv_draw_sw_mask_line_param_t *param, int32_t p1x, int32_t p1y, int32_t p2x, int32_t p2y, *lv_draw_sw_mask_line_side_t* side)

Initialize a line mask from two points.

Parametri

- **param** -- pointer to a `lv_draw_mask_param_t` to initialize
- **p1x** -- X coordinate of the first point of the line
- **p1y** -- Y coordinate of the first point of the line
- **p2x** -- X coordinate of the second point of the line
- **p2y** -- y coordinate of the second point of the line
- **side** -- and element of `lv_draw_mask_line_side_t` to describe which side to keep. With `LV_DRAW_MASK_LINE_SIDE_LEFT/RIGHT` and horizontal

line all pixels are kept With `LV_DRAW_MASK_LINE_SIDE_TOP/BOTTOM` and vertical line all pixels are kept

void **lv_draw_sw_mask_line_angle_init**(lv_draw_sw_mask_line_param_t *param, int32_t px, int32_t py, int16_t angle, *lv_draw_sw_mask_line_side_t* side)

Initialize a line mask from a point and an angle.

Parametri

- **param** -- pointer to a `lv_draw_mask_param_t` to initialize
- **px** -- X coordinate of a point of the line
- **py** -- X coordinate of a point of the line
- **angle** -- right 0 deg, bottom: 90
- **side** -- an element of `lv_draw_mask_line_side_t` to describe which side to keep. With `LV_DRAW_MASK_LINE_SIDE_LEFT/RIGHT` and horizontal line all pixels are kept With `LV_DRAW_MASK_LINE_SIDE_TOP/BOTTOM` and vertical line all pixels are kept

void **lv_draw_sw_mask_angle_init**(lv_draw_sw_mask_angle_param_t *param, int32_t vertex_x, int32_t vertex_y, int32_t start_angle, int32_t end_angle)

Initialize an angle mask.

Parametri

- **param** -- pointer to a `lv_draw_mask_param_t` to initialize
- **vertex_x** -- X coordinate of the angle vertex (absolute coordinates)
- **vertex_y** -- Y coordinate of the angle vertex (absolute coordinates)
- **start_angle** -- start angle in degrees. 0 deg on the right, 90 deg, on the bottom
- **end_angle** -- end angle

void **lv_draw_sw_mask_radius_init**(lv_draw_sw_mask_radius_param_t *param, const *lv_area_t* *rect, int32_t radius, bool inv)

Initialize a fade mask.

Parametri

- **param** -- pointer to an `lv_draw_mask_radius_param_t` to initialize
- **rect** -- coordinates of the rectangle to affect (absolute coordinates)
- **radius** -- radius of the rectangle
- **inv** -- true: keep the pixels inside the rectangle; keep the pixels outside of the rectangle

void **lv_draw_sw_mask_fade_init**(lv_draw_sw_mask_fade_param_t *param, const *lv_area_t* *coords, lv_opa_t opa_top, int32_t y_top, lv_opa_t opa_bottom, int32_t y_bottom)

Initialize a fade mask.

Parametri

- **param** -- pointer to a `lv_draw_mask_param_t` to initialize
- **coords** -- coordinates of the area to affect (absolute coordinates)
- **opa_top** -- opacity on the top
- **y_top** -- at which coordinate start to change to opacity to `opa_bottom`
- **opa_bottom** -- opacity at the bottom
- **y_bottom** -- at which coordinate reach `opa_bottom`.

void **lv_draw_sw_mask_map_init**(lv_draw_sw_mask_map_param_t *param, const *lv_area_t* *coords, const lv_opa_t *map)

Initialize a map mask.

Parametri

- **param** -- pointer to a `lv_draw_mask_param_t` to initialize
- **coords** -- coordinates of the map (absolute coordinates)
- **map** -- array of bytes with the mask values

lv_draw_sw_mask_private.h

Typedef

typedef *lv_draw_sw_mask_radius_circle_dsc_t* **lv_draw_sw_mask_radius_circle_dsc_arr_t**[4]

Functions

void **lv_draw_sw_mask_cleanup**(void)

Called by LVGL the rendering of a screen is ready to clean up the temporal (cache) data of the masks

struct **lv_draw_sw_mask_radius_circle_dsc_t**

Public Members

uint8_t ***buf**

lv_opa_t ***cir_opa**

Opacity of values on the circumference of an 1/4 circle

uint16_t ***x_start_on_y**

The x coordinate of the circle for each y value

uint16_t ***opa_start_on_y**

The index of `cir_opa` for each y value

int32_t **life**

How many times the entry way used

uint32_t **used_cnt**

Like a semaphore to count the referencing masks

int32_t **radius**

The radius of the entry

struct **_lv_draw_sw_mask_common_dsc_t**

Public Members

lv_draw_sw_mask_xcb_t **cb**

lv_draw_sw_mask_type_t **type**

struct **_lv_draw_sw_mask_line_param_t**

Public Members

lv_draw_sw_mask_common_dsc_t **dsc**

The first element must be the common descriptor

lv_point_t **p1**

lv_point_t **p2**

lv_draw_sw_mask_line_side_t **side**

struct *_lv_draw_sw_mask_line_param_t* **cfg**

lv_point_t **origo**

A point of the line

int32_t **xy_steep**

X / (1024*Y) steepness (X is 0..1023 range). What is the change of X in 1024 Y?

int32_t **yx_steep**

Y / (1024*X) steepness (Y is 0..1023 range). What is the change of Y in 1024 X?

int32_t **steep**

Helper which stores yx_steep for flat lines and xy_steep for steep (non flat) lines

int32_t **spx**

Steepness in 1 px in 0..255 range. Used only by flat lines.

uint8_t **flat**

1: It's a flat line? (Near to horizontal)

uint8_t **inv**

Invert the mask. The default is: Keep the left part. It is used to select left/right/top/bottom

struct **_lv_draw_sw_mask_angle_param_t**

Public Members

lv_draw_sw_mask_common_dsc_t **dsc**

The first element must be the common descriptor

lv_point_t **vertex_p**

int32_t **start_angle**

int32_t **end_angle**

struct *_lv_draw_sw_mask_angle_param_t* **cfg**

lv_draw_sw_mask_line_param_t **start_line**

lv_draw_sw_mask_line_param_t **end_line**

uint16_t **delta_deg**

struct **_lv_draw_sw_mask_radius_param_t**

Public Members

lv_draw_sw_mask_common_dsc_t **dsc**

The first element must be the common descriptor

lv_area_t **rect**

int32_t **radius**

uint8_t **outer**

Invert the mask. 0: Keep the pixels inside.

struct *_lv_draw_sw_mask_radius_param_t* **cfg**

lv_draw_sw_mask_radius_circle_dsc_t ***circle**

struct **_lv_draw_sw_mask_fade_param_t**

Public Members

lv_draw_sw_mask_common_dsc_t **dsc**

The first element must be the common descriptor

lv_area_t **coords**

int32_t **y_top**

int32_t **y_bottom**

lv_opa_t **opa_top**

lv_opa_t **opa_bottom**

struct *_lv_draw_sw_mask_fade_param_t* **cfg**

struct **_lv_draw_sw_mask_map_param_t**

Public Members

lv_draw_sw_mask_common_dsc_t **dsc**

The first element must be the common descriptor

lv_area_t **coords**

const lv_opa_t ***map**

struct *_lv_draw_sw_mask_map_param_t* **cfg**

lv_draw_sw_private.h

struct **lv_draw_sw_thread_dsc_t**

Public Members

lv_draw_task_t ***task_act**

lv_thread_t **thread**

lv_thread_sync_t **sync**

lv_draw_unit_t ***draw_unit**

uint32_t **idx**

volatile bool **inited**

volatile bool **exit_status**

struct **_lv_draw_sw_unit_t**

Public Members

lv_draw_unit_t **base_unit**

lv_draw_task_t ***task_act**

lv_draw_sw_utils.h

Functions

void **lv_draw_sw_i1_to_argb8888**(const void *buf_i1, void *buf_argb8888, uint32_t width, uint32_t height, uint32_t buf_i1_stride, uint32_t buf_argb8888_stride, uint32_t index0_color, uint32_t index1_color)

Converts an I1 buffer to ARGB8888 format.

Parametri

- **buf_i1** -- pointer to buffer with I1 formatted render
- **buf_argb8888** -- pointer to buffer for ARGB8888 render
- **width** -- width in pixels of the area. must be a multiple of 8.
- **height** -- height in pixels of the area
- **buf_i1_stride** -- stride of i1 buffer in bytes
- **buf_argb8888_stride** -- stride of argb8888 buffer in bytes
- **index0_color** -- color of the 0 bits of i1 buf

- **index1_color** -- color of the 1 bits of i1 buf

void **lv_draw_sw_rgb565_swap**(void *buf, uint32_t buf_size_px)

Swap the upper and lower byte of an RGB565 buffer. Might be required if a 8bit parallel port or an SPI port send the bytes in the wrong order. The bytes will be swapped in place.

Parametri

- **buf** -- pointer to buffer
- **buf_size_px** -- number of pixels in the buffer

void **lv_draw_sw_i1_invert**(void *buf, uint32_t buf_size)

Invert a draw buffer in the I1 color format. Conventionally, a bit is set to 1 during blending if the luminance is greater than 127. Depending on the display controller used, you might want to have different behavior. The inversion will be performed in place.

Parametri

- **buf** -- pointer to the buffer to be inverted
- **buf_size** -- size of the buffer in bytes

void **lv_draw_sw_i1_convert_to_vtiled**(const void *buf, uint32_t buf_size, uint32_t width, uint32_t height, void *out_buf, uint32_t out_buf_size, bool bit_order_lsb)

Convert a draw buffer in I1 color format from htiled (row-wise) to vtiled (column-wise) buffer layout. The conversion assumes that the buffer width and height is rounded to a multiple of 8.

Parametri

- **buf** -- pointer to the buffer to be converted
- **buf_size** -- size of the buffer in bytes
- **width** -- width of the buffer
- **height** -- height of the buffer
- **out_buf** -- pointer to the output buffer
- **out_buf_size** -- size of the output buffer in bytes
- **bit_order_lsb** -- bit order of the resulting vtiled buffer

void **lv_draw_sw_rotate**(const void *src, void *dest, int32_t src_width, int32_t src_height, int32_t src_stride, int32_t dest_stride, *lv_display_rotation_t* rotation, *lv_color_format_t* color_format)

Rotate a buffer into another buffer

Parametri

- **src** -- the source buffer
- **dest** -- the destination buffer
- **src_width** -- source width in pixels
- **src_height** -- source height in pixels
- **src_stride** -- source stride in bytes (number of bytes in a row)
- **dest_stride** -- destination stride in bytes (number of bytes in a row)
- **rotation** -- LV_DISPLAY_ROTATION_0/90/180/270
- **color_format** -- LV_COLOR_FORMAT_RGB565/RGB888/XRGB8888/ARGB8888

arm2d

lv_draw_sw_arm2d.h

lv_draw_sw_helium.h

blend

lv_draw_sw_blend.h

Typedef

typedef void (***lv_draw_sw_blend_handler_t**)(lv_draw_task_t *t, const lv_draw_sw_blend_dsc_t *dsc)

Custom draw function for SW rendering.

Param t
pointer to a draw task
Param dsc
pointer to an initialized blend descriptor

Functions

void **lv_draw_sw_blend**(lv_draw_task_t *t, const lv_draw_sw_blend_dsc_t *dsc)

Call the blend function of the `layer`.

Parametri

- **draw_unit** -- pointer to a draw unit
- **dsc** -- pointer to an initialized blend descriptor

struct **lv_draw_sw_custom_blend_handler_t**

Public Members

lv_color_format_t **dest_cf**

lv_draw_sw_blend_handler_t **handler**

lv_draw_sw_blend_private.h

struct **_lv_draw_sw_blend_dsc_t**

Public Members

const *lv_area_t* ***blend_area**

The area with absolute coordinates to draw on `layer->buf` will be clipped to `layer->clip_area`

const void ***src_buf**

Pointer to an image to blend. If set `fill_color` is ignored

uint32_t **src_stride**

lv_color_format_t **src_color_format**

const *lv_area_t* ***src_area**

lv_opa_t **opa**

The overall opacity

lv_color_t **color**

Fill color

const lv_opa_t ***mask_buf**

NULL if ignored, or an alpha mask to apply on `blend_area`

lv_draw_sw_mask_res_t **mask_res**

The result of the previous mask operation

const *lv_area_t* ***mask_area***

The area of `mask_buf` with absolute coordinates

int32_t **mask_stride**

lv_blend_mode_t **blend_mode**

E.g. LV_BLEND_MODE_ADDITIVE

struct **_lv_draw_sw_blend_fill_dsc_t**

Public Members

void ***dest_buf**

int32_t **dest_w**

int32_t **dest_h**

int32_t **dest_stride**

const lv_opa_t ***mask_buf**

int32_t **mask_stride**

lv_color_t **color**

lv_opa_t **opa**

lv_area_t **relative_area**

struct **_lv_draw_sw_blend_image_dsc_t**

Public Members

void ***dest_buf**

int32_t **dest_w**

int32_t **dest_h**

int32_t **dest_stride**

const lv_opa_t ***mask_buf**

int32_t **mask_stride**

const void ***src_buf**

int32_t **src_stride**

lv_color_format_t **src_color_format**

lv_opa_t **opa**

lv_blend_mode_t **blend_mode**

lv_area_t **relative_area**

The blend area relative to the layer's buffer area.

lv_area_t **src_area**

The original src area.

lv_draw_sw_blend_to_al88.h

Functions

void **lv_draw_sw_blend_color_to_al88**(lv_draw_sw_blend_fill_dsc_t *dsc)

void **lv_draw_sw_blend_image_to_al88**(lv_draw_sw_blend_image_dsc_t *dsc)

lv_draw_sw_blend_to_argb8888.h

Functions

void **lv_draw_sw_blend_color_to_argb8888**(lv_draw_sw_blend_fill_dsc_t *dsc)

void **lv_draw_sw_blend_image_to_argb8888**(lv_draw_sw_blend_image_dsc_t *dsc)

lv_draw_sw_blend_to_argb8888_premultiplied.h

Functions

void **lv_draw_sw_blend_color_to_argb8888_premultiplied**(lv_draw_sw_blend_fill_dsc_t *dsc)

void **lv_draw_sw_blend_image_to_argb8888_premultiplied**(lv_draw_sw_blend_image_dsc_t *dsc)

lv_draw_sw_blend_to_i1.h

Functions

void **lv_draw_sw_blend_color_to_i1**(lv_draw_sw_blend_fill_dsc_t *dsc)

void **lv_draw_sw_blend_image_to_i1**(lv_draw_sw_blend_image_dsc_t *dsc)

lv_draw_sw_blend_to_l8.h

Functions

void **lv_draw_sw_blend_color_to_l8**(lv_draw_sw_blend_fill_dsc_t *dsc)

void **lv_draw_sw_blend_image_to_l8**(lv_draw_sw_blend_image_dsc_t *dsc)

lv_draw_sw_blend_to_rgb565.h

Functions

void **lv_draw_sw_blend_color_to_rgb565**(lv_draw_sw_blend_fill_dsc_t *dsc)

void **lv_draw_sw_blend_image_to_rgb565**(lv_draw_sw_blend_image_dsc_t *dsc)

lv_draw_sw_blend_to_rgb565_swapped.h

Functions

void **lv_draw_sw_blend_color_to_rgb565_swapped**(lv_draw_sw_blend_fill_dsc_t *dsc)

void **lv_draw_sw_blend_image_to_rgb565_swapped**(lv_draw_sw_blend_image_dsc_t *dsc)

lv_draw_sw_blend_to_rgb888.h

Functions

void **lv_draw_sw_blend_color_to_rgb888**(lv_draw_sw_blend_fill_dsc_t *dsc, uint32_t dest_px_size)

void **lv_draw_sw_blend_image_to_rgb888**(lv_draw_sw_blend_image_dsc_t *dsc, uint32_t dest_px_size)

arm2d

lv_blend_arm2d.h

helium

lv_blend_helium.h

neon

lv_blend_neon.h

14.12.32 vg_lite

lv_draw_vg_lite.h

Functions

void **lv_draw_buf_vg_lite_init_handlers**(void)

void **lv_draw_vg_lite_init**(void)

void **lv_draw_vg_lite_deinit**(void)

void **lv_draw_vg_lite_arc**(lv_draw_task_t *t, const *lv_draw_arc_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_vg_lite_box_shadow**(lv_draw_task_t *t, const *lv_draw_box_shadow_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_vg_lite_border**(lv_draw_task_t *t, const *lv_draw_border_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_vg_lite_fill**(lv_draw_task_t *t, const *lv_draw_fill_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_vg_lite_img**(lv_draw_task_t *t, const lv_draw_image_dsc_t *dsc, const *lv_area_t* *coords, bool no_cache)

void **lv_draw_vg_lite_label_init**(struct *_lv_draw_vg_lite_unit_t* *u)

void **lv_draw_vg_lite_label_deinit**(struct *_lv_draw_vg_lite_unit_t* *u)

void **lv_draw_vg_lite_letter**(lv_draw_task_t *t, const *lv_draw_letter_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_vg_lite_label**(lv_draw_task_t *t, const *lv_draw_label_dsc_t* *dsc, const *lv_area_t* *coords)

void **lv_draw_vg_lite_layer**(lv_draw_task_t *t, const lv_draw_image_dsc_t *draw_dsc, const *lv_area_t* *coords)

void **lv_draw_vg_lite_line**(lv_draw_task_t *t, const *lv_draw_line_dsc_t* *dsc)

void **lv_draw_vg_lite_triangle**(lv_draw_task_t *t, const *lv_draw_triangle_dsc_t* *dsc)

void **lv_draw_vg_lite_mask_rect**(lv_draw_task_t *t, const lv_draw_mask_rect_dsc_t *dsc, const *lv_area_t* *coords)

void **lv_draw_vg_lite_vector**(lv_draw_task_t *t, const lv_draw_vector_task_dsc_t *dsc)

lv_draw_vg_lite_type.h

struct **_lv_draw_vg_lite_unit_t**

Public Members

lv_draw_unit_t **base_unit**

lv_draw_task_t ***task_act**

struct _lv_vg_lite_pending_t ***image_dsc_pending**

struct _lv_vg_lite_grad_ctx_t ***grad_ctx**

lv_cache_t ***stroke_cache**

struct _lv_vg_lite_pending_t ***bitmap_font_pending**

uint16_t **flush_count**

uint16_t **letter_count**

vg_lite_buffer_t **target_buffer**

vg_lite_matrix_t **global_matrix**

struct _lv_vg_lite_path_t ***global_path**

bool **path_in_use**

lv_vg_lite_decoder.h

Functions

void **lv_vg_lite_decoder_init**(void)

void **lv_vg_lite_decoder_deinit**(void)

lv_vg_lite_grad.h

Functions

struct _lv_vg_lite_grad_ctx_t ***lv_vg_lite_grad_ctx_create**(uint32_t cache_cnt, struct *_lv_draw_vg_lite_unit_t* *unit)

Create a gradient context.

Parametri

- **cache_cnt** -- number of cache entries
- **unit** -- the draw unit

void **lv_vg_lite_grad_ctx_delete**(struct _lv_vg_lite_grad_ctx_t *ctx)

Delete a gradient context.

Parametri

ctx -- the gradient context to delete

struct _lv_vg_lite_pending_t ***lv_vg_lite_grad_ctx_get_pending**(struct _lv_vg_lite_grad_ctx_t *ctx)

Get the pending list of gradient items.

Parametri

ctx -- the gradient context

bool **lv_vg_lite_draw_grad**(struct _lv_vg_lite_grad_ctx_t *ctx, vg_lite_buffer_t *buffer, vg_lite_path_t *path, const lv_vector_gradient_t *grad, const vg_lite_matrix_t *grad_matrix, const vg_lite_matrix_t *matrix, vg_lite_fill_t fill, vg_lite_blend_t blend)

Draw a gradient.

Parametri

- **ctx** -- the gradient context
- **buffer** -- the target buffer
- **path** -- the path to draw the gradient on
- **grad** -- the gradient descriptor
- **grad_matrix** -- the gradient matrix
- **matrix** -- the matrix to apply to the gradient
- **fill** -- the fill rule
- **blend** -- the blend mode

Ritorna

true: success, false: failed

bool **lv_vg_lite_draw_grad_helper**(struct _lv_vg_lite_grad_ctx_t *ctx, vg_lite_buffer_t *buffer, vg_lite_path_t *path, const *lv_area_t* *area, const *lv_grad_dsc_t* *grad_dsc, const vg_lite_matrix_t *matrix, vg_lite_fill_t fill, vg_lite_blend_t blend)

Draw a gradient helper.

Parametri

- **ctx** -- the gradient context
- **buffer** -- the target buffer
- **path** -- the path to draw the gradient on
- **area** -- the area to draw the gradient on
- **grad_dsc** -- the gradient descriptor
- **matrix** -- the matrix to apply to the gradient
- **fill** -- the fill rule
- **blend** -- the blend mode

Ritorna

true: success, false: failed

lv_vg_lite_math.h

Defines

MATH_PI

MATH_HALF_PI

MATH_TWO_PI

DEG_TO_RAD

RAD_TO_DEG

MATH_TANF(x)

MATH_SINF(x)

MATH_COSF(x)

MATH_ASINF(x)

MATH_ACOSF(x)

MATH_FABSF(x)

MATH_SQRTF(x)

MATH_RADIANS(deg)

MATH_DEGREES(rad)

Functions

static inline bool **math_zero**(float a)

static inline bool **math_equal**(float a, float b)

float **math_fast_inv_sqrtf**(float number)

lv_vg_lite_path.h

Defines

PATH_COORD_MAX

It is found that thorvg cannot handle large coordinates well. When the coordinates are larger than 4096, the calculation of tvgSwRle module will overflow in 32-bit system. So we use FLT_MAX and FLT_MIN to write the mark to bounding_box to tell vg_lite_tvg not to add clip path to the current path.

PATH_COORD_MIN

LV_VG_LITE_PATH_SET_OP_CODE(PTR, TYPE, OP_CODE)

LV_VG_LITE_PATH_GET_OP_CODE(PTR)

Typedef

typedef struct _lv_vg_lite_path_t **lv_vg_lite_path_t**

typedef struct *_lv_draw_vg_lite_unit_t* **lv_draw_vg_lite_unit_t**

typedef void (***lv_vg_lite_path_iter_cb_t**)(void *user_data, uint8_t op_code, const float *data, uint32_t len)

Functions

void **lv_vg_lite_path_init**(*lv_draw_vg_lite_unit_t* *unit)

void **lv_vg_lite_path_deinit**(*lv_draw_vg_lite_unit_t* *unit)

lv_vg_lite_path_t ***lv_vg_lite_path_create**(vg_lite_format_t data_format)

void **lv_vg_lite_path_destroy**(*lv_vg_lite_path_t* *path)

lv_vg_lite_path_t ***lv_vg_lite_path_get**(*lv_draw_vg_lite_unit_t* *unit, vg_lite_format_t data_format)

void **lv_vg_lite_path_drop**(*lv_draw_vg_lite_unit_t* *unit, *lv_vg_lite_path_t* *path)

void **lv_vg_lite_path_reset**(*lv_vg_lite_path_t* *path, vg_lite_format_t data_format)

void **lv_vg_lite_path_set_bounding_box_area**(*lv_vg_lite_path_t* *path, const *lv_area_t* *area)

void **lv_vg_lite_path_set_bounding_box**(*lv_vg_lite_path_t* *path, float min_x, float min_y, float max_x, float max_y)

void **lv_vg_lite_path_get_bounding_box**(*lv_vg_lite_path_t* *path, float *min_x, float *min_y, float *max_x, float *max_y)

bool **lv_vg_lite_path_update_bounding_box**(*lv_vg_lite_path_t* *path)

void **lv_vg_lite_path_set_transform**(*lv_vg_lite_path_t* *path, const vg_lite_matrix_t *matrix)

void **lv_vg_lite_path_set_quality**(*lv_vg_lite_path_t* *path, vg_lite_quality_t quality)

vg_lite_path_t ***lv_vg_lite_path_get_path**(*lv_vg_lite_path_t* *path)

void **lv_vg_lite_path_reserve_space**(*lv_vg_lite_path_t* *path, size_t len)

void **lv_vg_lite_path_move_to**(*lv_vg_lite_path_t* *path, float x, float y)

void **lv_vg_lite_path_line_to**(*lv_vg_lite_path_t* *path, float x, float y)

void **lv_vg_lite_path_quad_to**(*lv_vg_lite_path_t* *path, float cx, float cy, float x, float y)

void **lv_vg_lite_path_cubic_to**(*lv_vg_lite_path_t* *path, float cx1, float cy1, float cx2, float cy2, float x, float y)

void **lv_vg_lite_path_close**(*lv_vg_lite_path_t* *path)

void **lv_vg_lite_path_end**(*lv_vg_lite_path_t* *path)

void **lv_vg_lite_path_append_rect**(*lv_vg_lite_path_t* *path, float x, float y, float w, float h, float r)

void **lv_vg_lite_path_append_circle**(*lv_vg_lite_path_t* *path, float cx, float cy, float rx, float ry)

void **lv_vg_lite_path_append_arc_right_angle**(*lv_vg_lite_path_t* *path, float start_x, float start_y, float center_x, float center_y, float end_x, float end_y)

void **lv_vg_lite_path_append_arc**(*lv_vg_lite_path_t* *path, float cx, float cy, float radius, float start_angle, float sweep, bool pie)

void **lv_vg_lite_path_append_path**(*lv_vg_lite_path_t* *dest, const *lv_vg_lite_path_t* *src)

uint8_t **lv_vg_lite_vlc_op_arg_len**(uint8_t vlc_op)

uint8_t **lv_vg_lite_path_format_len**(vg_lite_format_t format)

void **lv_vg_lite_path_for_each_data**(const vg_lite_path_t *path, *lv_vg_lite_path_iter_cb_t* cb, void *user_data)

lv_vg_lite_pending.h

Typedef

typedef struct _lv_vg_lite_pending_t **lv_vg_lite_pending_t**

typedef void (***lv_vg_lite_pending_free_cb_t**)(void *obj, void *user_data)

Functions

lv_vg_lite_pending_t ***lv_vg_lite_pending_create**(size_t obj_size, uint32_t capacity_default)

Create a pending list

Parametri

- **obj_size** -- the size of the objects in the list
- **capacity_default** -- the default capacity of the list

Ritorna

a pointer to the pending list

void **lv_vg_lite_pending_destroy**(*lv_vg_lite_pending_t* *pending)

Destroy a pending list

Parametri

pending -- pointer to the pending list

void **lv_vg_lite_pending_set_free_cb**(*lv_vg_lite_pending_t* *pending, *lv_vg_lite_pending_free_cb_t* free_cb, void *user_data)

Set a free callback for the pending list

Parametri

- **pending** -- pointer to the pending list
- **free_cb** -- the free callback
- **user_data** -- user data to pass to the free callback

void **lv_vg_lite_pending_add**(*lv_vg_lite_pending_t* *pending, void *obj)

Add an object to the pending list

Parametri

- **pending** -- pointer to the pending list
- **obj** -- pointer to the object to add

void **lv_vg_lite_pending_remove_all**(*lv_vg_lite_pending_t* *pending)

Remove all objects from the active pending list

Parametri

pending -- pointer to the pending list

void **lv_vg_lite_pending_swap**(*lv_vg_lite_pending_t* *pending)

Remove all old objects reference and swap new objects reference

Parametri

pending -- pointer to the pending list

lv_vg_lite_stroke.h

Functions

void **lv_vg_lite_stroke_init**(struct *_lv_draw_vg_lite_unit_t* *unit, uint32_t cache_cnt)

Initialize the stroke module

Parametri

unit -- pointer to the unit

void **lv_vg_lite_stroke_deinit**(struct *_lv_draw_vg_lite_unit_t* *unit)

Deinitialize the stroke module

Parametri

unit -- pointer to the unit

lv_cache_entry_t ***lv_vg_lite_stroke_get**(struct *_lv_draw_vg_lite_unit_t* *unit, struct _lv_vg_lite_path_t *path, const lv_vector_stroke_dsc_t *dsc)

Get the stroke cache entry

Parametri

- **unit** -- pointer to the unit
- **path** -- pointer to the path
- **dsc** -- pointer to the stroke descriptor

Ritorna

pointer to the stroke cache entry

struct _lv_vg_lite_path_t ***lv_vg_lite_stroke_get_path**(lv_cache_entry_t *cache_entry)

Get the path of a stroke

Parametri

cache_entry -- pointer to the stroke cache entry

Ritorna

pointer to the path

void **lv_vg_lite_stroke_drop**(struct *_lv_draw_vg_lite_unit_t* *unit, lv_cache_entry_t *cache_entry)

Drop the stroke cache entry

Parametri

- **unit** -- pointer to the unit
- **stroke** -- pointer to the stroke

lv_vg_lite_utils.h

Defines

LV_VG_LITE_ASSERT(expr)

LV_VG_LITE_CHECK_ERROR(expr, error_handler)

LV_VG_LITE_ASSERT_PATH(path)

LV_VG_LITE_ASSERT_SRC_BUFFER(buffer)

LV_VG_LITE_ASSERT_DEST_BUFFER(buffer)

LV_VG_LITE_ASSERT_MATRIX(matrix)

LV_VG_LITE_ALIGN(number, align_bytes)

LV_VG_LITE_IS_ALIGNED(num, align)

LV_VG_LITE_IS_INDEX_FMT(fmt)

Functions

void **lv_vg_lite_dump_info**(void)

void **lv_vg_lite_error_dump_info**(vg_lite_error_t error)

const char ***lv_vg_lite_error_string**(vg_lite_error_t error)

const char ***lv_vg_lite_feature_string**(vg_lite_feature_t feature)

const char ***lv_vg_lite_buffer_format_string**(vg_lite_buffer_format_t format)

const char ***lv_vg_lite_vlc_op_string**(uint8_t vlc_op)

void **lv_vg_lite_path_dump_info**(const vg_lite_path_t *path)

void **lv_vg_lite_stroke_dump_info**(const vg_lite_stroke_t *stroke)

void **lv_vg_lite_buffer_dump_info**(const vg_lite_buffer_t *buffer)

void **lv_vg_lite_matrix_dump_info**(const vg_lite_matrix_t *matrix)

void **lv_vg_lite_color_dump_info**(const vg_lite_color_t color)

bool **lv_vg_lite_is_dest_cf_supported**(*lv_color_format_t* cf)

bool **lv_vg_lite_is_src_cf_supported**(*lv_color_format_t* cf)

vg_lite_buffer_format_t **lv_vg_lite_vg_fmt**(*lv_color_format_t* cf)

void **lv_vg_lite_buffer_format_bytes**(vg_lite_buffer_format_t format, uint32_t *mul, uint32_t *div, uint32_t *bytes_align)

uint32_t **lv_vg_lite_width_to_stride**(uint32_t w, vg_lite_buffer_format_t color_format)

uint32_t **lv_vg_lite_width_align**(uint32_t w)

void **lv_vg_lite_buffer_init**(vg_lite_buffer_t *buffer, const void *ptr, int32_t width, int32_t height, uint32_t stride, vg_lite_buffer_format_t format, bool tiled)

void **lv_vg_lite_buffer_from_draw_buf**(vg_lite_buffer_t *buffer, const lv_draw_buf_t *draw_buf)

void **lv_vg_lite_image_matrix**(vg_lite_matrix_t *matrix, int32_t x, int32_t y, const lv_draw_image_dsc_t *dsc)

vg_lite_color_t **lv_vg_lite_image_recolor**(vg_lite_buffer_t *buffer, const lv_draw_image_dsc_t *dsc)

bool **lv_vg_lite_buffer_open_image**(vg_lite_buffer_t *buffer, lv_image_decoder_dsc_t *decoder_dsc, const void *src, bool no_cache, bool premultiply)

void **lv_vg_lite_image_dsc_init**(struct *_lv_draw_vg_lite_unit_t* *unit)

void **lv_vg_lite_image_dsc_deinit**(struct *_lv_draw_vg_lite_unit_t* *unit)

vg_lite_blend_t **lv_vg_lite_blend_mode**(*lv_blend_mode_t* blend_mode, bool has_pre_mul)

uint32_t **lv_vg_lite_get_palette_size**(vg_lite_buffer_format_t format)

vg_lite_color_t **lv_vg_lite_color**(*lv_color_t* color, lv_opa_t opa, bool pre_mul)

void **lv_vg_lite_rect**(vg_lite_rectangle_t *rect, const *lv_area_t* *area)

static inline void **lv_vg_lite_matrix**(vg_lite_matrix_t *dest, const lv_matrix_t *src)

bool **lv_vg_lite_buffer_check**(const vg_lite_buffer_t *buffer, bool is_src)

bool **lv_vg_lite_path_check**(const vg_lite_path_t *path)

bool **lv_vg_lite_matrix_check**(const vg_lite_matrix_t *matrix)

bool **lv_vg_lite_support_blend_normal**(void)

bool **lv_vg_lite_16px_align**(void)

void **lv_vg_lite_matrix_multiply**(vg_lite_matrix_t *matrix, const vg_lite_matrix_t *mult)

bool **lv_vg_lite_matrix_inverse**(vg_lite_matrix_t *result, const vg_lite_matrix_t *matrix)

lv_point_precise_t **lv_vg_lite_matrix_transform_point**(const vg_lite_matrix_t *matrix, const *lv_point_precise_t* *point)

void **lv_vg_lite_set_scissor_area**(const *lv_area_t* *area)

void **lv_vg_lite_disable_scissor**(void)

void **lv_vg_lite_flush**(struct *_lv_draw_vg_lite_unit_t* *u)

void **lv_vg_lite_finish**(struct *_lv_draw_vg_lite_unit_t* *u)

static inline void **lv_vg_lite_draw**(vg_lite_buffer_t *target, vg_lite_path_t *path, vg_lite_fill_t fill_rule, vg_lite_matrix_t *matrix, vg_lite_blend_t blend, vg_lite_color_t color)

static inline void **lv_vg_lite_draw_pattern**(vg_lite_buffer_t *target, vg_lite_path_t *path, vg_lite_fill_t fill_rule, vg_lite_matrix_t *path_matrix, vg_lite_buffer_t *pattern_image, vg_lite_matrix_t *pattern_matrix, vg_lite_blend_t blend, vg_lite_pattern_mode_t pattern_mode, vg_lite_color_t pattern_color, vg_lite_color_t color, vg_lite_filter_t filter)

static inline void **lv_vg_lite_blit_rect**(vg_lite_buffer_t *target, vg_lite_buffer_t *source, vg_lite_rectangle_t *rect, vg_lite_matrix_t *matrix, vg_lite_blend_t blend, vg_lite_color_t color, vg_lite_filter_t filter)

static inline void **lv_vg_lite_clear**(vg_lite_buffer_t *target, const *lv_area_t* *area, vg_lite_color_t color)

14.13 drivers

14.13.1 lv_drivers.h

14.13.2 display

drm

lv_linux_drm.h

Functions

lv_display_t ***lv_linux_drm_create**(void)

void **lv_linux_drm_set_file**(lv_display_t *disp, const char *file, int64_t connector_id)

fb

lv_linux_fbdev.h

Functions

lv_display_t ***lv_linux_fbdev_create**(void)

void **lv_linux_fbdev_set_file**(lv_display_t *disp, const char *file)

void **lv_linux_fbdev_set_force_refresh**(lv_display_t *disp, bool enabled)

Force the display to be refreshed on every change. Expected to be used with LV_DISPLAY_RENDER_MODE_DIRECT or LV_DISPLAY_RENDER_MODE_FULL.

ft81x

lv_ft81x.h

Typedef

typedef void (***lv_ft81x_spi_cb_t**)(lv_display_t *disp, *lv_ft81x_spi_operation_t* operation, void *data, uint32_t length)

Enums

enum **lv_ft81x_spi_operation_t**

Values:

enumerator **LV_FT81X_SPI_OPERATION_CS_ASSERT**

enumerator **LV_FT81X_SPI_OPERATION_CS_DEASSERT**

enumerator **LV_FT81X_SPI_OPERATION_SEND**

enumerator **LV_FT81X_SPI_OPERATION_RECEIVE**

Functions

lv_display_t ***lv_ft81x_create**(const *lv_ft81x_parameters_t* *params, void *partial_buf, uint32_t buf_size, *lv_ft81x_spi_cb_t* spi_cb, void *user_data)

Create a framebuffer-based ft81x driver display.

Parametri

- **params** -- pointer to a struct of display panel properties. does not need to be static.
- **partial_buf** -- a single partial buffer
- **buf_size** -- size of the partial buffer
- **spi_cb** -- a callback called by the driver to perform SPI operations
- **user_data** -- use `lv_ft81x_get_user_data` to get this pointer inside the SPI callback

Ritorna

pointer to the display

void ***lv_ft81x_get_user_data**(lv_display_t *disp)

Get the `user_data` parameter that was passed to `lv_ft81x_create`. Useful in the SPI callback.

Parametri

disp -- pointer to the ft81x display

Ritorna

the `user_data` pointer

struct **lv_ft81x_parameters_t**

Public Members

uint16_t **hor_res**

active display width

uint16_t **ver_res**

active display height

uint16_t **hcycle**

total number of clocks per line, incl front/back porch

uint16_t **hoffset**

start of active line

uint16_t **hsync0**

start of horizontal sync pulse

uint16_t **hsync1**

end of horizontal sync pulse

uint16_t **vcycle**

total number of lines per screen, including pre/post

uint16_t **voffset**

start of active screen

uint16_t **vsync0**

start of vertical sync pulse

uint16_t **vsync1**

end of vertical sync pulse

uint8_t **swizzle**

FT8xx output to LCD - pin order

uint8_t **pclkpol**

LCD data is clocked in on this PCLK edge

uint8_t **cspread**

helps with noise, when set to 1 fewer signals are changed simultaneously, reset-default: 1

uint8_t **pclk**

60MHz / pclk = pclk frequency

bool **has_crystal**

has an external clock crystal

bool **is_bt81x**

is a BT series model, not FT

lv_ft81x_defines.h

Copied from https://github.com/lvgl/lvgl_esp32_drivers/blob/9fed1cc47b5a45fec6bae08b55d2147d3b50260c/lvgl_tft/EVE.h

Defines

FT81X_ENABLE

DL_CLEAR

DL_CLEAR_RGB

DL_COLOR_RGB

DL_POINT_SIZE

DL_END

DL_BEGIN

DL_DISPLAY

CLR_COL

CLR_STN

CLR_TAG

SPI_WIDTH_SIO

SPI_WIDTH_DIO

SPI_WIDTH_QIO

EVE_ACTIVE

EVE_STANDBY

EVE_SLEEP

EVE_PWRDOWN

EVE_CLKEXT

EVE_CLKINT

EVE_CORERST

EVE_CLK48M

EVE_CLK36M

EVE_NEVER

EVE_LESS

EVE_LEQUAL

EVE_GREATER

EVE_GEQUAL

EVE_EQUAL

EVE_NOTEQUAL

EVE_ALWAYS

EVE_ARGB1555

EVE_L1

EVE_L4

EVE_L8

EVE_RGB332

EVE_ARGB2

EVE_ARGB4

EVE_RGB565

EVE_PALETTED

EVE_TEXT8X8

EVE_TEXTVGA

EVE_BARGRAPH

EVE_NEAREST

EVE_BILINEAR

EVE_BORDER

EVE_REPEAT

EVE_KEEP

EVE_REPLACE

EVE_INCR

EVE_DECR

EVE_INVERT

EVE_DLSWAP_DONE

EVE_DLSWAP_LINE

EVE_DLSWAP_FRAME

EVE_INT_SWAP

EVE_INT_TOUCH

EVE_INT_TAG

EVE_INT_SOUND

EVE_INT_PLAYBACK

EVE_INT_CMDEMPTY

EVE_INT_CMDFLAG

EVE_INT_CONVCOMPLETE

EVE_TMODE_OFF

EVE_TMODE_ONESHOT

EVE_TMODE_FRAME

EVE_TMODE_CONTINUOUS

EVE_ZERO

EVE_ONE

EVE_SRC_ALPHA

EVE_DST_ALPHA

EVE_ONE_MINUS_SRC_ALPHA

EVE_ONE_MINUS_DST_ALPHA

EVE_BITMAPS

EVE_POINTS

EVE_LINES

EVE_LINE_STRIP

EVE_EDGE_STRIP_R

EVE_EDGE_STRIP_L

EVE_EDGE_STRIP_A

EVE_EDGE_STRIP_B

EVE_RECTS

EVE_OPT_MONO

EVE_OPT_NODL

EVE_OPT_FLAT

EVE_OPT_CENTERX

EVE_OPT_CENTERY

EVE_OPT_CENTER

EVE_OPT_NOBACK

EVE_OPT_NOTICKS

EVE_OPT_NOHM

EVE_OPT_NOPOINTER

EVE_OPT_NOSECS

EVE_OPT_NOHANDS

EVE_OPT_RIGHTX

EVE_OPT_SIGNED

EVE_NUMCHAR_PERFONT

EVE_FONT_TABLE_SIZE

EVE_FONT_TABLE_POINTER

EVE_LINEAR_SAMPLES

EVE_ULAW_SAMPLES

EVE_ADPCM_SAMPLES

EVE_SILENCE

EVE_SQUAREWAVE

EVE_SINEWAVE

EVE_SAWTOOTH

EVE_TRIANGLE

EVE_BEEPING

EVE_ALARM

EVE_WARBLE

EVE_CAROUSEL

EVE_PIPS(n)

EVE_HARP

EVE_XYLOPHONE

EVE_TUBA

EVE_GLOCKENSPIEL

EVE_ORGAN

EVE_TRUMPET

EVE_PIANO

EVE_CHIMES

EVE_MUSICBOX

EVE_BELL

EVE_CLICK

EVE_SWITCH

EVE_COWBELL

EVE_NOTCH

EVE_HIHAT

EVE_KICKDRUM

EVE_POP

EVE_CLACK

EVE_CHACK

EVE_MUTE

EVE_UNMUTE

EVE_MIDI_A0

EVE_MIDI_A_0

EVE_MIDI_B0

EVE_MIDI_C1

EVE_MIDI_C_1

EVE_MIDI_D1

EVE_MIDI_D_1

EVE_MIDI_E1

EVE_MIDI_F1

EVE_MIDI_F_1

EVE_MIDI_G1

EVE_MIDI_G_1

EVE_MIDI_A1

EVE_MIDI_A_1

EVE_MIDI_B1

EVE_MIDI_C2

EVE_MIDI_C_2

EVE_MIDI_D2

EVE_MIDI_D_2

EVE_MIDI_E2

EVE_MIDI_F2

EVE_MIDI_F_2

EVE_MIDI_G2

EVE_MIDI_G_2

EVE_MIDI_A2

EVE_MIDI_A_2

EVE_MIDI_B2

EVE_MIDI_C3

EVE_MIDI_C_3

EVE_MIDI_D3

EVE_MIDI_D_3

EVE_MIDI_E3

EVE_MIDI_F3

EVE_MIDI_F_3

EVE_MIDI_G3

EVE_MIDI_G_3

EVE_MIDI_A3

EVE_MIDI_A_3

EVE_MIDI_B3

EVE_MIDI_C4

EVE_MIDI_C_4

EVE_MIDI_D4

EVE_MIDI_D_4

EVE_MIDI_E4

EVE_MIDI_F4

EVE_MIDI_F_4

EVE_MIDI_G4

EVE_MIDI_G_4

EVE_MIDI_A4

EVE_MIDI_A_4

EVE_MIDI_B4

EVE_MIDI_C5

EVE_MIDI_C_5

EVE_MIDI_D5

EVE_MIDI_D_5

EVE_MIDI_E5

EVE_MIDI_F5

EVE_MIDI_F_5

EVE_MIDI_G5

EVE_MIDI_G_5

EVE_MIDI_A5

EVE_MIDI_A_5

EVE_MIDI_B5

EVE_MIDI_C6

EVE_MIDI_C_6

EVE_MIDI_D6

EVE_MIDI_D_6

EVE_MIDI_E6

EVE_MIDI_F6

EVE_MIDI_F_6

EVE_MIDI_G6

EVE_MIDI_G_6

EVE_MIDI_A6

EVE_MIDI_A_6

EVE_MIDI_B6

EVE_MIDI_C7

EVE_MIDI_C_7

EVE_MIDI_D7

EVE_MIDI_D_7

EVE_MIDI_E7

EVE_MIDI_F7

EVE_MIDI_F_7

EVE_MIDI_G7

EVE_MIDI_G_7

EVE_MIDI_A7

EVE_MIDI_A_7

EVE_MIDI_B7

EVE_MIDI_C8

EVE_GPIO0

EVE_GPIO1

EVE_GPIO7

EVE_DISPLAY_0

EVE_DISPLAY_180

CMD_APPEND

CMD_BGCOLOR

CMD_BUTTON

CMD_CALIBRATE

CMD_CLOCK

CMD_COLDSTART

CMD_DIAL

CMD_DLSTART

CMD_FGCOLOR

CMD_GAUGE

CMD_GETMATRIX

CMD_GETPROPS

CMD_GETPTR

CMD_GRADCOLOR

CMD_GRADIENT

CMD_INFLATE

CMD_INTERRUPT

CMD_KEYS

CMD_LOADIDENTITY

CMD_LOADIMAGE

CMD_LOGO

CMD_MEMCPY

CMD_MEMCRC

CMD_MEMSET

CMD_MEMWRITE

CMD_MEMZERO

CMD_NUMBER

CMD_PROGRESS

CMD_REGREAD

CMD_ROTATE

CMD_SCALE

CMD_SCREENSAVER

CMD_SCROLLBAR

CMD_SETFONT

CMD_SETMATRIX

CMD_SKETCH

CMD_SLIDER

CMD_SNAPSHOT

CMD_SPINNER

CMD_STOP

CMD_SWAP

CMD_TEXT

CMD_TOGGLE

CMD_TRACK

CMD_TRANSLATE

ALPHA_FUNC(func, ref)

BEGIN(prim)

BITMAP_HANDLE(handle)

BITMAP_LAYOUT(format, linestride, height)

BITMAP_SIZE(filter, wrapx, wrapy, width, height)

BITMAP_TRANSFORM_A(a)

BITMAP_TRANSFORM_B(b)

BITMAP_TRANSFORM_C(c)

BITMAP_TRANSFORM_D(d)

BITMAP_TRANSFORM_E(e)

BITMAP_TRANSFORM_F(f)

BLEND_FUNC(src, dst)

CALL(dest)

CELL(cell)

CLEAR(c, s, t)

CLEAR_COLOR_A(alpha)

CLEAR_COLOR_RGB(red, green, blue)

CLEAR_STENCIL(s)

CLEAR_TAG(s)

COLOR_A(alpha)

COLOR_MASK(r, g, b, a)

COLOR_RGB(red, green, blue)

END()

JUMP(dest)

LINE_WIDTH(width)

MACRO(m)

POINT_SIZE(size)

RESTORE_CONTEXT()

RETURN()

SAVE_CONTEXT()

STENCIL_FUNC(func, ref, mask)

STENCIL_MASK(mask)

STENCIL_OP(sfail, spass)

TAG(s)

TAG_MASK(mask)

VERTEX2F(x, y)

VERTEX2II(x, y, handle, cell)

EVE_CLKSEL

EVE_RST_PULSE

EVE_PINDRIVE

EVE_PIN_PD_STATE

EVE_RAM_G

EVE_ROM_CHIPID

EVE_ROM_FONT

EVE_ROM_FONT_ADDR

EVE_RAM_DL

EVE_RAM_REG

EVE_RAM_CMD

EVE_RAM_G_SIZE

EVE_CMDFIFO_SIZE

EVE_RAM_DL_SIZE

EVE_ADC_DIFFERENTIAL

EVE_ADC_SINGLE_ENDED

EVE_INT_G8

EVE_INT_L8C

EVE_INT_VGA

EVE_OPT_MEDIAFIFO

EVE_OPT_FULLSCREEN

EVE_OPT_NOTEAR

EVE_OPT_SOUND

EVE_PALETTED565

EVE_PALETTED4444

EVE_PALETTED8

EVE_L2

CMD_MEDIAFIFO

CMD_PLAYVIDEO

CMD_ROMFONT

CMD_SETBASE

CMD_SETBITMAP

CMD_SETFONT2

CMD_SETROTATE

CMD_SETSCRATCH

CMD_SNAPSHOT2

CMD_VIDEOFRAME

CMD_VIDEOSTART

REG_ANA_COMP

REG_BIST_EN

REG_CLOCK

REG_CMDB_SPACE

REG_CMDB_WRITE

REG_CMD_DL

REG_CMD_READ

REG_CMD_WRITE

REG_CPURESET

REG_CSPREAD

REG_CTOUCH_EXTENDED

REG_CTOUCH_TOUCH0_XY

REG_CTOUCH_TOUCH4_X

REG_CTOUCH_TOUCH4_Y

REG_CTOUCH_TOUCH1_XY

REG_CTOUCH_TOUCH2_XY

REG_CTOUCH_TOUCH3_XY

REG_TOUCH_CONFIG

REG_DATESTAMP

REG_DITHER

REG_DLSWAP

REG_FRAMES

REG_FREQUENCY

REG_GPIO

REG_GPIOX

REG_GPIOX_DIR

REG_GPIO_DIR

REG_HCYCLE

REG_HOFFSET

REG_HSIZE

REG_HSYNC0

REG_HSYNC1

REG_ID

REG_INT_EN

REG_INT_FLAGS

REG_INT_MASK

REG_MACRO_0

REG_MACRO_1

REG_MEDIAFIFO_READ

REG_MEDIAFIFO_WRITE

REG_OUTBITS

REG_PCLK

REG_PCLK_POL

REG_PLAY

REG_PLAYBACK_FORMAT

REG_PLAYBACK_FREQ

REG_PLAYBACK_LENGTH

REG_PLAYBACK_LOOP

REG_PLAYBACK_PLAY

REG_PLAYBACK_READPTR

REG_PLAYBACK_START

REG_PWM_DUTY

REG_PWM_HZ

REG_RENDERMODE

REG_ROTATE

REG_SNAPFORMAT

REG_SNAPSHOT

REG_SNAPY

REG_SOUND

REG_SPI_WIDTH

REG_SWIZZLE

REG_TAG

REG_TAG_X

REG_TAG_Y

REG_TAP_CRC

REG_TAP_MASK

REG_TOUCH_ADC_MODE

REG_TOUCH_CHARGE

REG_TOUCH_DIRECT_XY

REG_TOUCH_DIRECT_Z1Z2

REG_TOUCH_MODE

REG_TOUCH_OVERSAMPLE

REG_TOUCH_RAW_XY

REG_TOUCH_RZ

REG_TOUCH_RZTHRESH

REG_TOUCH_SCREEN_XY

REG_TOUCH_SETTLE

REG_TOUCH_TAG

REG_TOUCH_TAG1

REG_TOUCH_TAG1_XY

REG_TOUCH_TAG2

REG_TOUCH_TAG2_XY

REG_TOUCH_TAG3

REG_TOUCH_TAG3_XY

REG_TOUCH_TAG4

REG_TOUCH_TAG4_XY

REG_TOUCH_TAG_XY

REG_TOUCH_TRANSFORM_A

REG_TOUCH_TRANSFORM_B

REG_TOUCH_TRANSFORM_C

REG_TOUCH_TRANSFORM_D

REG_TOUCH_TRANSFORM_E

REG_TOUCH_TRANSFORM_F

REG_TRACKER

REG_TRACKER_1

REG_TRACKER_2

REG_TRACKER_3

REG_TRACKER_4

REG_TRIM

REG_VCYCLE

REG_VOFFSET

REG_VOL_PB

REG_VOL_SOUND

REG_VSIZE

REG_VSYNC0

REG_VSYNC1

BITMAP_LAYOUT_H(linestride, height)

BITMAP_SIZE_H(width, height)

BITMAP_SOURCE(addr)

PALETTE_SOURCE(addr)

SCISSOR_SIZE(width, height)

SCISSOR_XY(x, y)

VERTEX_FORMAT(frac)

VERTEX_TRANSLATE_X(x)

VERTEX_TRANSLATE_Y(y)

ili9341

lv_ili9341.h

This driver is just a wrapper around the generic MIPI compatible LCD controller driver

Typedef

typedef *lv_lcd_send_cmd_cb_t* **lv_ili9341_send_cmd_cb_t**

typedef *lv_lcd_send_color_cb_t* **lv_ili9341_send_color_cb_t**

Functions

lv_display_t ***lv_ili9341_create**(uint32_t hor_res, uint32_t ver_res, *lv_lcd_flag_t* flags, *lv_ili9341_send_cmd_cb_t* send_cmd_cb, *lv_ili9341_send_color_cb_t* send_color_cb)

Create an LCD display with ILI9341 driver

Parametri

- **hor_res** -- horizontal resolution
- **ver_res** -- vertical resolution
- **flags** -- default configuration settings (mirror, RGB ordering, etc.)
- **send_cmd** -- platform-dependent function to send a command to the LCD controller (usually uses polling transfer)

- **send_color** -- platform-dependent function to send pixel data to the LCD controller (usually uses DMA transfer: must implement a 'ready' callback)

Ritorna

pointer to the created display

void **lv_ili9341_set_gap**(lv_display_t *disp, uint16_t x, uint16_t y)

Set gap, i.e., the offset of the (0,0) pixel in the VRAM

Parametri

- **disp** -- display object
- **x** -- x offset
- **y** -- y offset

void **lv_ili9341_set_invert**(lv_display_t *disp, bool invert)

Set color inversion

Parametri

- **disp** -- display object
- **invert** -- false: normal, true: invert

void **lv_ili9341_set_gamma_curve**(lv_display_t *disp, uint8_t gamma)

Set gamma curve

Parametri

- **disp** -- display object
- **gamma** -- gamma curve

void **lv_ili9341_send_cmd_list**(lv_display_t *disp, const uint8_t *cmd_list)

Send list of commands.

Parametri

- **disp** -- display object
- **cmd_list** -- controller and panel-specific commands

lcd

lv_lcd_generic_mipi.h

Generic driver for controllers adhering to the MIPI DBI/DCS specification

Works with:

ST7735 ST7789 ST7796 ILI9341 ILI9488 (NOTE: in SPI mode ILI9488 only supports RGB666 mode, which is currently not supported)

any probably many more

Defines

LV_LCD_CMD_NOP

LV_LCD_CMD_SOFT_RESET

LV_LCD_CMD_GET_POWER_MODE

LV_LCD_CMD_GET_ADDRESS_MODE

LV_LCD_CMD_GET_PIXEL_FORMAT

LV_LCD_CMD_GET_DISPLAY_MODE

LV_LCD_CMD_GET_SIGNAL_MODE

LV_LCD_CMD_GET_DIAGNOSTIC_RESULT

LV_LCD_CMD_ENTER_SLEEP_MODE

LV_LCD_CMD_EXIT_SLEEP_MODE

LV_LCD_CMD_ENTER_PARTIAL_MODE

LV_LCD_CMD_ENTER_NORMAL_MODE

LV_LCD_CMD_EXIT_INVERT_MODE

LV_LCD_CMD_ENTER_INVERT_MODE

LV_LCD_CMD_SET_GAMMA_CURVE

LV_LCD_CMD_SET_DISPLAY_OFF

LV_LCD_CMD_SET_DISPLAY_ON

LV_LCD_CMD_SET_COLUMN_ADDRESS

LV_LCD_CMD_SET_PAGE_ADDRESS

LV_LCD_CMD_WRITE_MEMORY_START

LV_LCD_CMD_READ_MEMORY_START

LV_LCD_CMD_SET_PARTIAL_ROWS

LV_LCD_CMD_SET_PARTIAL_COLUMNS

LV_LCD_CMD_SET_SCROLL_AREA

LV_LCD_CMD_SET_TEAR_OFF

LV_LCD_CMD_SET_TEAR_ON

LV_LCD_CMD_SET_ADDRESS_MODE

LV_LCD_CMD_SET_SCROLL_START

LV_LCD_CMD_EXIT_IDLE_MODE

LV_LCD_CMD_ENTER_IDLE_MODE

LV_LCD_CMD_SET_PIXEL_FORMAT

LV_LCD_CMD_WRITE_MEMORY_CONTINUE

LV_LCD_CMD_READ_MEMORY_CONTINUE

LV_LCD_CMD_SET_TEAR_SCANLINE

LV_LCD_CMD_GET_SCANLINE

LV_LCD_CMD_READ_DDB_CONTINUE

LV_LCD_CMD_READ_DDB_START

LV_LCD_MASK_FLIP_VERTICAL

LV_LCD_MASK_FLIP_HORIZONTAL

LV_LCD_MASK_DATA_LATCH_DATA_ORDER

LV_LCD_MASK_RGB_ORDER

LV_LCD_MASK_LINE_ADDRESS_ORDER

LV_LCD_MASK_PAGE_COLUMN_ORDER

LV_LCD_MASK_COLUMN_ADDRESS_ORDER

LV_LCD_MASK_PAGE_ADDRESS_ORDER

LV_LCD_BIT_FLIP_VERTICAL__NOT_FLIPPED

LV_LCD_BIT_FLIP_VERTICAL__FLIPPED

LV_LCD_BIT_FLIP_HORIZONTAL__NOT_FLIPPED

LV_LCD_BIT_FLIP_HORIZONTAL__FLIPPED

LV_LCD_BIT_DATA_LATCH_DATA_ORDER__LTOR

LV_LCD_BIT_DATA_LATCH_DATA_ORDER__RTOL

LV_LCD_BIT_RGB_ORDER__RGB

LV_LCD_BIT_RGB_ORDER__BGR

LV_LCD_BIT_LINE_ADDRESS_ORDER__TTOB

LV_LCD_BIT_LINE_ADDRESS_ORDER__BTOT

LV_LCD_BIT_PAGE_COLUMN_ORDER__NORMAL

LV_LCD_BIT_PAGE_COLUMN_ORDER__REVERSE

LV_LCD_BIT_COLUMN_ADDRESS_ORDER__LTOR

LV_LCD_BIT_COLUMN_ADDRESS_ORDER__RTOL

LV_LCD_BIT_PAGE_ADDRESS_ORDER__TTOB

LV_LCD_BIT_PAGE_ADDRESS_ORDER__BTOT

LV_LCD_GAMMA_2_2

LV_LCD_GAMMA_1_8

LV_LCD_GAMMA_2_5

LV_LCD_GAMMA_1_0

LV_LCD_PIXEL_FORMAT_RGB565

LV_LCD_PIXEL_FORMAT_RGB666

LV_LCD_FLAG_NONE

LV_LCD_FLAG_MIRROR_X

LV_LCD_FLAG_MIRROR_Y

LV_LCD_FLAG_BGR

LV_LCD_FLAG_RGB666

LV_LCD_CMD_DELAY_MS

LV_LCD_CMD_EOF

Typedef

typedef uint32_t **lv_lcd_flag_t**

Configuration flags for lv_lcd_xxx_create()

typedef void (***lv_lcd_send_cmd_cb_t**)(lv_display_t *disp, const uint8_t *cmd, size_t cmd_size, const uint8_t *param, size_t param_size)

Prototype of a platform-dependent callback to transfer commands and data to the LCD controller.

Param disp
display object

Param cmd
command buffer (can handle 16 bit commands as well)

Param cmd_size
number of bytes of the command

Param param
parameter buffer

Param param_size
number of bytes of the parameters

typedef void (***lv_lcd_send_color_cb_t**)(lv_display_t *disp, const uint8_t *cmd, size_t cmd_size, uint8_t *param, size_t param_size)

Prototype of a platform-dependent callback to transfer pixel data to the LCD controller.

Param disp
display object

Param cmd
command buffer (can handle 16 bit commands as well)

Param cmd_size
number of bytes of the command

Param param
parameter buffer

Param param_size
number of bytes of the parameters

Functions

lv_display_t ***lv_lcd_generic_mipi_create**(uint32_t hor_res, uint32_t ver_res, *lv_lcd_flag_t* flags, *lv_lcd_send_cmd_cb_t* send_cmd_cb, *lv_lcd_send_color_cb_t* send_color_cb)

Create a MIPI DCS compatible LCD display

Parametri

- **hor_res** -- horizontal resolution
- **ver_res** -- vertical resolution
- **flags** -- default configuration settings (mirror, RGB ordering, etc.)
- **send_cmd** -- platform-dependent function to send a command to the LCD controller (usually uses polling transfer)
- **send_color** -- platform-dependent function to send pixel data to the LCD controller (usually uses DMA transfer). `lv_display_flush_ready` must be called after the transfer has finished.

Ritorna
pointer to the created display

void **lv_lcd_generic_mipi_set_gap**(lv_display_t *disp, uint16_t x, uint16_t y)

Set gap, i.e., the offset of the (0,0) pixel in the VRAM

Parametri

- **disp** -- display object
- **x** -- x offset
- **y** -- y offset

void **lv_lcd_generic_mipi_set_invert**(lv_display_t *disp, bool invert)

Set color inversion

Parametri

- **disp** -- display object
- **invert** -- false: normal, true: invert

void **lv_lcd_generic_mipi_set_address_mode**(lv_display_t *disp, bool mirror_x, bool mirror_y, bool swap_xy, bool bgr)

Set address mode

Parametri

- **disp** -- display object
- **mirror_x** -- horizontal mirror (false: normal, true: mirrored)
- **mirror_y** -- vertical mirror (false: normal, true: mirrored)
- **swap_xy** -- swap axes (false: normal, true: swap)
- **bgr** -- RGB/BGR order (false: RGB, true: BGR)

void **lv_lcd_generic_mipi_set_gamma_curve**(lv_display_t *disp, uint8_t gamma)

Set gamma curve

Parametri

- **disp** -- display object
- **gamma** -- gamma curve

void **lv_lcd_generic_mipi_send_cmd_list**(lv_display_t *disp, const uint8_t *cmd_list)

Send list of commands.

Parametri

- **disp** -- display object
- **cmd_list** -- controller and panel-specific commands

struct **lv_lcd_generic_mipi_driver_t**

#include <lv_lcd_generic_mipi.h> Generic MIPI compatible LCD driver

Public Members

lv_display_t ***disp**

lv_lcd_send_cmd_cb_t **send_cmd**

lv_lcd_send_color_cb_t **send_color**

uint16_t **x_gap**

uint16_t **y_gap**

uint8_t **madctl_reg**

uint8_t **colmod_reg**

bool **mirror_x**

bool **mirror_y**

bool **swap_xy**

nxp_elcdif

lv_nxp_elcdif.h

Driver for NXP's ELCD

Functions

lv_display_t ***lv_nxp_display_elcdif_create_direct**(LCDIF_Type *base, const elcdif_rgb_mode_config_t *config, void *frame_buffer1, void *frame_buffer2, size_t buf_size)

Attach LVGL to ELCDIF using DIRECT rendering mode. ELCDIF should be already initialized.

Parametri

- **base** -- The NXP eLCD controller base address
- **config** -- NXP eLCD config object
- **frame_buffer1** -- pointer the first frame buffers
- **frame_buffer2** -- pointer the second frame buffers
- **buf_size** -- size of a buffer in bytes (must be at least as large as the screen)

Ritorna

a display object initialized and registerd on the LVGL runtime

lv_display_t ***lv_nxp_display_elcdif_create_partial**(LCDIF_Type *base, const elcdif_rgb_mode_config_t *config, void *frame_buffer1, void *frame_buffer2, size_t buf_size)

Attach LVGL to ELCDIF using PARTIAL rendering mode. ELCDIF should be already initialized.

Parametri

- **base** -- The NXP eLCD controller base address
- **config** -- NXP eLCD config object
- **frame_buffer1** -- pointer the first frame buffers
- **frame_buffer2** -- pointer the second frame buffers
- **buf_size** -- size of a buffer in bytes

Ritorna

a display object initialized and registerd on the LVGL runtime

void **lv_nxp_display_elcdif_event_handler**(const lv_display_t *disp)

Call this function on the LCD Interrupt Service Routine It tells to LVGL what to do when a framebuffer is transmitted to the LCD panel

> **Nota**
>
> : the parameter disp is tipycally the return value after `lv_nxp_display_elcdif_create_direct` has been sucessfully executed

Parametri

disp -- The display instance that contains the eLCD related data

renesas_glcdc

lv_renesas_glcdc.h

Functions

lv_display_t ***lv_renesas_glcdc_direct_create**(void)

Create a display using Renesas' GLCDC peripheral in DIRECT render mode

Ritorna

pointer to the created display

lv_display_t *`lv_renesas_glcdc_partial_create`(void *buf1, void *buf2, size_t buf_size)

Create a display using Renesas' GLCDC peripheral in PARTIAL render mode

Parametri

- **buf1** -- first buffer
- **buf2** -- second buffer (can be NULL)
- **buf_size** -- buffer size in byte

Ritorna

pointer to the created display

st7735

lv_st7735.h

This driver is just a wrapper around the generic MIPI compatible LCD controller driver

Typedef

typedef *lv_lcd_send_cmd_cb_t* `lv_st7735_send_cmd_cb_t`

typedef *lv_lcd_send_color_cb_t* `lv_st7735_send_color_cb_t`

Functions

lv_display_t *`lv_st7735_create`(uint32_t hor_res, uint32_t ver_res, *lv_lcd_flag_t* flags, *lv_st7735_send_cmd_cb_t* send_cmd_cb, *lv_st7735_send_color_cb_t* send_color_cb)

Create an LCD display with ST7735 driver

Parametri

- **hor_res** -- horizontal resolution
- **ver_res** -- vertical resolution
- **flags** -- default configuration settings (mirror, RGB ordering, etc.)
- **send_cmd** -- platform-dependent function to send a command to the LCD controller (usually uses polling transfer)
- **send_color** -- platform-dependent function to send pixel data to the LCD controller (usually uses DMA transfer: must implement a 'ready' callback)

Ritorna

pointer to the created display

void `lv_st7735_set_gap`(lv_display_t *disp, uint16_t x, uint16_t y)

Set gap, i.e., the offset of the (0,0) pixel in the VRAM

Parametri

- **disp** -- display object
- **x** -- x offset
- **y** -- y offset

void `lv_st7735_set_invert`(lv_display_t *disp, bool invert)

Set color inversion

Parametri

- **disp** -- display object
- **invert** -- false: normal, true: invert

void `lv_st7735_set_gamma_curve`(lv_display_t *disp, uint8_t gamma)

Set gamma curve

Parametri

- **disp** -- display object
- **gamma** -- gamma curve

void **lv_st7735_send_cmd_list**(lv_display_t *disp, const uint8_t *cmd_list)

Send list of commands.

Parametri

- **disp** -- display object
- **cmd_list** -- controller and panel-specific commands

st7789

lv_st7789.h

This driver is just a wrapper around the generic MIPI compatible LCD controller driver

Typedef

typedef *lv_lcd_send_cmd_cb_t* **lv_st7789_send_cmd_cb_t**

typedef *lv_lcd_send_color_cb_t* **lv_st7789_send_color_cb_t**

Functions

lv_display_t ***lv_st7789_create**(uint32_t hor_res, uint32_t ver_res, *lv_lcd_flag_t* flags, *lv_st7789_send_cmd_cb_t* send_cmd_cb, *lv_st7789_send_color_cb_t* send_color_cb)

Create an LCD display with ST7789 driver

Parametri

- **hor_res** -- horizontal resolution
- **ver_res** -- vertical resolution
- **flags** -- default configuration settings (mirror, RGB ordering, etc.)
- **send_cmd** -- platform-dependent function to send a command to the LCD controller (usually uses polling transfer)
- **send_color** -- platform-dependent function to send pixel data to the LCD controller (usually uses DMA transfer: must implement a 'ready' callback)

Ritorna

pointer to the created display

void **lv_st7789_set_gap**(lv_display_t *disp, uint16_t x, uint16_t y)

Set gap, i.e., the offset of the (0,0) pixel in the VRAM

Parametri

- **disp** -- display object
- **x** -- x offset
- **y** -- y offset

void **lv_st7789_set_invert**(lv_display_t *disp, bool invert)

Set color inversion

Parametri

- **disp** -- display object
- **invert** -- false: normal, true: invert

void **lv_st7789_set_gamma_curve**(lv_display_t *disp, uint8_t gamma)

Set gamma curve

Parametri

- **disp** -- display object
- **gamma** -- gamma curve

void **lv_st7789_send_cmd_list**(lv_display_t *disp, const uint8_t *cmd_list)

Send list of commands.

Parametri

- **disp** -- display object

- **cmd_list** -- controller and panel-specific commands

st7796

lv_st7796.h

This driver is just a wrapper around the generic MIPI compatible LCD controller driver

Typedef

typedef *lv_lcd_send_cmd_cb_t* **lv_st7796_send_cmd_cb_t**

typedef *lv_lcd_send_color_cb_t* **lv_st7796_send_color_cb_t**

Functions

lv_display_t ***lv_st7796_create**(uint32_t hor_res, uint32_t ver_res, *lv_lcd_flag_t* flags, *lv_st7796_send_cmd_cb_t* send_cmd_cb, *lv_st7796_send_color_cb_t* send_color_cb)

Create an LCD display with ST7796 driver

Parametri

- **hor_res** -- horizontal resolution
- **ver_res** -- vertical resolution
- **flags** -- default configuration settings (mirror, RGB ordering, etc.)
- **send_cmd** -- platform-dependent function to send a command to the LCD controller (usually uses polling transfer)
- **send_color** -- platform-dependent function to send pixel data to the LCD controller (usually uses DMA transfer: must implement a 'ready' callback)

Ritorna

pointer to the created display

void **lv_st7796_set_gap**(lv_display_t *disp, uint16_t x, uint16_t y)

Set gap, i.e., the offset of the (0,0) pixel in the VRAM

Parametri

- **disp** -- display object
- **x** -- x offset
- **y** -- y offset

void **lv_st7796_set_invert**(lv_display_t *disp, bool invert)

Set color inversion

Parametri

- **disp** -- display object
- **invert** -- false: normal, true: invert

void **lv_st7796_set_gamma_curve**(lv_display_t *disp, uint8_t gamma)

Set gamma curve

Parametri

- **disp** -- display object
- **gamma** -- gamma curve

void **lv_st7796_send_cmd_list**(lv_display_t *disp, const uint8_t *cmd_list)

Send list of commands.

Parametri

- **disp** -- display object
- **cmd_list** -- controller and panel-specific commands

st_ltdc

lv_st_ltdc.h

Functions

lv_display_t *`lv_st_ltdc_create_direct`(void *fb_adr_1, void *fb_adr_2, uint32_t layer_idx)

Create a direct render mode display bound to a LTDC layer.

Parametri

- **fb_adr_1** -- The LTDC layer's framebuffer memory address.
- **fb_adr_2** -- An additional framebuffer-sized buffer to use for double buffering, or `NULL`.
- **layer_idx** -- The LTDC layer number to bind the display to. Typically 0 or 1.

Ritorna

The display.

lv_display_t *`lv_st_ltdc_create_partial`(void *render_buf_1, void *render_buf_2, uint32_t buf_size, uint32_t layer_idx)

Create a partial render mode display bound to a LTDC layer. The layer's framebuffer is flushed to internally. Enable `LV_ST_LTDC_USE_DMA2D_FLUSH` for parallel flushing.

Parametri

- **render_buf_1** -- A render buffer.
- **render_buf_2** -- An additional render buffer for double-buffering, or `NULL`.
- **buf_size** -- The size of the buffer(s) in bytes.
- **layer_idx** -- The LTDC layer number to bind the display to. Typically 0 or 1.

Ritorna

The display.

tft_espi

lv_tft_espi.h

Functions

lv_display_t *`lv_tft_espi_create`(uint32_t hor_res, uint32_t ver_res, void *buf, uint32_t buf_size_bytes)

14.13.3 draw

eve

lv_draw_eve_display.h

Functions

lv_display_t *`lv_draw_eve_display_create`(const *lv_draw_eve_parameters_t* *params, *lv_draw_eve_operation_cb_t* op_cb, void *user_data)

Create a display for the EVE draw unit.

Parametri

- **params** -- Pointer to a struct of display parameters. Can be a temporary variable
- **op_cb** -- A callback that will be called to perform pin and SPI IO operations with the EVE chip
- **user_data** -- use `lv_draw_eve_display_get_user_data` to get this pointer inside the `op_cb`

Ritorna

the EVE display

void *`lv_draw_eve_display_get_user_data`(lv_display_t *disp)

Get the `user_data` parameter that was passed to `lv_draw_eve_display_create`. Useful in the operation callback.

Parametri

disp -- pointer to the lv_draw_eve display

Ritorna
the `user_data` pointer

lv_indev_t *`lv_draw_eve_touch_create`(lv_display_t *disp)

Create a touchscreen indev for the EVE display.

Parametri
`disp` -- pointer to the lv_draw_eve display

Ritorna
the EVE touchscreen indev

uint8_t `lv_draw_eve_memread8`(lv_display_t *disp, uint32_t address)

Call `EVE_memRead8` for custom low-level control of the display.

Parametri
- **`disp`** -- the display returned by `lv_draw_eve_display_create`
- **`address`** -- the EVE address to read from

Ritorna
the read value

uint16_t `lv_draw_eve_memread16`(lv_display_t *disp, uint32_t address)

Call `EVE_memRead16` for custom low-level control of the display.

Parametri
- **`disp`** -- the display returned by `lv_draw_eve_display_create`
- **`address`** -- the EVE address to read from

Ritorna
the read value

uint32_t `lv_draw_eve_memread32`(lv_display_t *disp, uint32_t address)

Call `EVE_memRead32` for custom low-level control of the display.

Parametri
- **`disp`** -- the display returned by `lv_draw_eve_display_create`
- **`address`** -- the EVE address to read from

Ritorna
the read value

void `lv_draw_eve_memwrite8`(lv_display_t *disp, uint32_t address, uint8_t data)

Call `EVE_memWrite8` for custom low-level control of the display.

Parametri
- **`disp`** -- the display returned by `lv_draw_eve_display_create`
- **`address`** -- the EVE address to write to
- **`data`** -- the value to write

void `lv_draw_eve_memwrite16`(lv_display_t *disp, uint32_t address, uint16_t data)

Call `EVE_memWrite16` for custom low-level control of the display.

Parametri
- **`disp`** -- the display returned by `lv_draw_eve_display_create`
- **`address`** -- the EVE address to write to
- **`data`** -- the value to write

void `lv_draw_eve_memwrite32`(lv_display_t *disp, uint32_t address, uint32_t data)

Call `EVE_memWrite32` for custom low-level control of the display.

Parametri
- **`disp`** -- the display returned by `lv_draw_eve_display_create`
- **`address`** -- the EVE address to write to
- **`data`** -- the value to write

lv_draw_eve_display_defines.h

Defines

LV_EVE_EVE_RAM_G

LV_EVE_EVE_ROM_CHIPID

LV_EVE_EVE_ROM_FONT

LV_EVE_EVE_ROM_FONTROOT

LV_EVE_EVE_RAM_DL

LV_EVE_EVE_RAM_REG

LV_EVE_EVE_RAM_CMD

LV_EVE_EVE_RAM_G_SIZE

LV_EVE_EVE_CMDFIFO_SIZE

LV_EVE_EVE_RAM_DL_SIZE

LV_EVE_DL_DISPLAY

LV_EVE_DL_BITMAP_SOURCE

LV_EVE_DL_CLEAR_COLOR_RGB

LV_EVE_DL_TAG

LV_EVE_DL_COLOR_RGB

LV_EVE_DL_BITMAP_HANDLE

LV_EVE_DL_CELL

LV_EVE_DL_BITMAP_LAYOUT

LV_EVE_DL_BITMAP_SIZE

LV_EVE_DL_ALPHA_FUNC

LV_EVE_DL_STENCIL_FUNC

LV_EVE_DL_BLEND_FUNC

LV_EVE_DL_STENCIL_OP

LV_EVE_DL_POINT_SIZE

LV_EVE_DL_LINE_WIDTH

LV_EVE_DL_CLEAR_COLOR_A

LV_EVE_DL_COLOR_A

LV_EVE_DL_CLEAR_STENCIL

LV_EVE_DL_CLEAR_TAG

LV_EVE_DL_STENCIL_MASK

LV_EVE_DL_TAG_MASK

LV_EVE_DL_BITMAP_TRANSFORM_A

LV_EVE_DL_BITMAP_TRANSFORM_B

LV_EVE_DL_BITMAP_TRANSFORM_C

LV_EVE_DL_BITMAP_TRANSFORM_D

LV_EVE_DL_BITMAP_TRANSFORM_E

LV_EVE_DL_BITMAP_TRANSFORM_F

LV_EVE_DL_SCISSOR_XY

LV_EVE_DL_SCISSOR_SIZE

LV_EVE_DL_CALL

LV_EVE_DL_JUMP

LV_EVE_DL_BEGIN

LV_EVE_DL_COLOR_MASK

LV_EVE_DL_END

LV_EVE_DL_SAVE_CONTEXT

LV_EVE_DL_RESTORE_CONTEXT

LV_EVE_DL_RETURN

LV_EVE_DL_MACRO

LV_EVE_DL_CLEAR

LV_EVE_DL_VERTEX_FORMAT

LV_EVE_DL_BITMAP_LAYOUT_H

LV_EVE_DL_BITMAP_SIZE_H

LV_EVE_DL_PALETTE_SOURCE

LV_EVE_DL_VERTEX_TRANSLATE_X

LV_EVE_DL_VERTEX_TRANSLATE_Y

LV_EVE_DL_NOP

LV_EVE_DL_VERTEX2F

LV_EVE_DL_VERTEX2II

LV_EVE_CLR_COL

LV_EVE_CLR_STN

LV_EVE_CLR_TAG

LV_EVE_EVE_ACTIVE

LV_EVE_EVE_STANDBY

LV_EVE_EVE_SLEEP

LV_EVE_EVE_CLKEXT

LV_EVE_EVE_CLKINT

LV_EVE_EVE_PWRDOWN

LV_EVE_EVE_CLKSEL

LV_EVE_EVE_RST_PULSE

LV_EVE_EVE_CORERST

LV_EVE_EVE_PINDRIVE

LV_EVE_EVE_PIN_PD_STATE

LV_EVE_EVE_NEVER

LV_EVE_EVE_LESS

LV_EVE_EVE_LEQUAL

LV_EVE_EVE_GREATER

LV_EVE_EVE_GEQUAL

LV_EVE_EVE_EQUAL

LV_EVE_EVE_NOTEQUAL

LV_EVE_EVE_ALWAYS

LV_EVE_EVE_ARGB1555

LV_EVE_EVE_L1

LV_EVE_EVE_L4

LV_EVE_EVE_L8

LV_EVE_EVE_RGB332

LV_EVE_EVE_ARGB2

LV_EVE_EVE_ARGB4

LV_EVE_EVE_RGB565

LV_EVE_EVE_PALETTED

LV_EVE_EVE_TEXT8X8

LV_EVE_EVE_TEXTVGA

LV_EVE_EVE_BARGRAPH

LV_EVE_EVE_NEAREST

LV_EVE_EVE_BILINEAR

LV_EVE_EVE_BORDER

LV_EVE_EVE_REPEAT

LV_EVE_EVE_KEEP

LV_EVE_EVE_REPLACE

LV_EVE_EVE_INCR

LV_EVE_EVE_DECR

LV_EVE_EVE_INVERT

LV_EVE_EVE_DLSWAP_DONE

LV_EVE_EVE_DLSWAP_LINE

LV_EVE_EVE_DLSWAP_FRAME

LV_EVE_EVE_INT_SWAP

LV_EVE_EVE_INT_TOUCH

LV_EVE_EVE_INT_TAG

LV_EVE_EVE_INT_SOUND

LV_EVE_EVE_INT_PLAYBACK

LV_EVE_EVE_INT_CMDEMPTY

LV_EVE_EVE_INT_CMDFLAG

LV_EVE_EVE_INT_CONVCOMPLETE

LV_EVE_EVE_TMODE_OFF

LV_EVE_EVE_TMODE_ONESHOT

LV_EVE_EVE_TMODE_FRAME

LV_EVE_EVE_TMODE_CONTINUOUS

LV_EVE_EVE_ZERO

LV_EVE_EVE_ONE

LV_EVE_EVE_SRC_ALPHA

LV_EVE_EVE_DST_ALPHA

LV_EVE_EVE_ONE_MINUS_SRC_ALPHA

LV_EVE_EVE_ONE_MINUS_DST_ALPHA

LV_EVE_EVE_BITMAPS

LV_EVE_EVE_POINTS

LV_EVE_EVE_LINES

LV_EVE_EVE_LINE_STRIP

LV_EVE_EVE_EDGE_STRIP_R

LV_EVE_EVE_EDGE_STRIP_L

LV_EVE_EVE_EDGE_STRIP_A

LV_EVE_EVE_EDGE_STRIP_B

LV_EVE_EVE_RECTS

LV_EVE_EVE_INT_G8

LV_EVE_EVE_INT_L8C

LV_EVE_EVE_INT_VGA

LV_EVE_EVE_PALETTED565

LV_EVE_EVE_PALETTED4444

LV_EVE_EVE_PALETTED8

LV_EVE_EVE_L2

LV_EVE_EVE_OPT_MONO

LV_EVE_EVE_OPT_NODL

LV_EVE_EVE_OPT_FLAT

LV_EVE_EVE_OPT_CENTERX

LV_EVE_EVE_OPT_CENTERY

LV_EVE_EVE_OPT_CENTER

LV_EVE_EVE_OPT_NOBACK

LV_EVE_EVE_OPT_NOTICKS

LV_EVE_EVE_OPT_NOHM

LV_EVE_EVE_OPT_NOPOINTER

LV_EVE_EVE_OPT_NOSECS

LV_EVE_EVE_OPT_NOHANDS

LV_EVE_EVE_OPT_RIGHTX

LV_EVE_EVE_OPT_SIGNED

LV_EVE_EVE_OPT_MEDIAFIFO

LV_EVE_EVE_OPT_FULLSCREEN

LV_EVE_EVE_OPT_NOTEAR

LV_EVE_EVE_OPT_SOUND

LV_EVE_EVE_ADC_DIFFERENTIAL

LV_EVE_EVE_ADC_SINGLE_ENDED

LV_EVE_EVE_NUMCHAR_PERFONT

LV_EVE_EVE_FONT_TABLE_SIZE

LV_EVE_EVE_FONT_TABLE_POINTER

LV_EVE_EVE_LINEAR_SAMPLES

LV_EVE_EVE_ULAW_SAMPLES

LV_EVE_EVE_ADPCM_SAMPLES

LV_EVE_EVE_SILENCE

LV_EVE_EVE_SQUAREWAVE

LV_EVE_EVE_SINEWAVE

LV_EVE_EVE_SAWTOOTH

LV_EVE_EVE_TRIANGLE

LV_EVE_EVE_BEEPING

LV_EVE_EVE_ALARM

LV_EVE_EVE_WARBLE

LV_EVE_EVE_CAROUSEL

LV_EVE_EVE_PIPS(n)

LV_EVE_EVE_HARP

LV_EVE_EVE_XYLOPHONE

LV_EVE_EVE_TUBA

LV_EVE_EVE_GLOCKENSPIEL

LV_EVE_EVE_ORGAN

LV_EVE_EVE_TRUMPET

LV_EVE_EVE_PIANO

LV_EVE_EVE_CHIMES

LV_EVE_EVE_MUSICBOX

LV_EVE_EVE_BELL

LV_EVE_EVE_CLICK

LV_EVE_EVE_SWITCH

LV_EVE_EVE_COWBELL

LV_EVE_EVE_NOTCH

LV_EVE_EVE_HIHAT

LV_EVE_EVE_KICKDRUM

LV_EVE_EVE_POP

LV_EVE_EVE_CLACK

LV_EVE_EVE_CHACK

LV_EVE_EVE_MUTE

LV_EVE_EVE_UNMUTE

LV_EVE_EVE_MIDI_A0

LV_EVE_EVE_MIDI_A_0

LV_EVE_EVE_MIDI_B0

LV_EVE_EVE_MIDI_C1

LV_EVE_EVE_MIDI_C_1

LV_EVE_EVE_MIDI_D1

LV_EVE_EVE_MIDI_D_1

LV_EVE_EVE_MIDI_E1

LV_EVE_EVE_MIDI_F1

LV_EVE_EVE_MIDI_F_1

LV_EVE_EVE_MIDI_G1

LV_EVE_EVE_MIDI_G_1

LV_EVE_EVE_MIDI_A1

LV_EVE_EVE_MIDI_A_1

LV_EVE_EVE_MIDI_B1

LV_EVE_EVE_MIDI_C2

LV_EVE_EVE_MIDI_C_2

LV_EVE_EVE_MIDI_D2

LV_EVE_EVE_MIDI_D_2

LV_EVE_EVE_MIDI_E2

LV_EVE_EVE_MIDI_F2

LV_EVE_EVE_MIDI_F_2

LV_EVE_EVE_MIDI_G2

LV_EVE_EVE_MIDI_G_2

LV_EVE_EVE_MIDI_A2

LV_EVE_EVE_MIDI_A_2

LV_EVE_EVE_MIDI_B2

LV_EVE_EVE_MIDI_C3

LV_EVE_EVE_MIDI_C_3

LV_EVE_EVE_MIDI_D3

LV_EVE_EVE_MIDI_D_3

LV_EVE_EVE_MIDI_E3

LV_EVE_EVE_MIDI_F3

LV_EVE_EVE_MIDI_F_3

LV_EVE_EVE_MIDI_G3

LV_EVE_EVE_MIDI_G_3

LV_EVE_EVE_MIDI_A3

LV_EVE_EVE_MIDI_A_3

LV_EVE_EVE_MIDI_B3

LV_EVE_EVE_MIDI_C4

LV_EVE_EVE_MIDI_C_4

LV_EVE_EVE_MIDI_D4

LV_EVE_EVE_MIDI_D_4

LV_EVE_EVE_MIDI_E4

LV_EVE_EVE_MIDI_F4

LV_EVE_EVE_MIDI_F_4

LV_EVE_EVE_MIDI_G4

LV_EVE_EVE_MIDI_G_4

LV_EVE_EVE_MIDI_A4

LV_EVE_EVE_MIDI_A_4

LV_EVE_EVE_MIDI_B4

LV_EVE_EVE_MIDI_C5

LV_EVE_EVE_MIDI_C_5

LV_EVE_EVE_MIDI_D5

LV_EVE_EVE_MIDI_D_5

LV_EVE_EVE_MIDI_E5

LV_EVE_EVE_MIDI_F5

LV_EVE_EVE_MIDI_F_5

LV_EVE_EVE_MIDI_G5

LV_EVE_EVE_MIDI_G_5

LV_EVE_EVE_MIDI_A5

LV_EVE_EVE_MIDI_A_5

LV_EVE_EVE_MIDI_B5

LV_EVE_EVE_MIDI_C6

LV_EVE_EVE_MIDI_C_6

LV_EVE_EVE_MIDI_D6

LV_EVE_EVE_MIDI_D_6

LV_EVE_EVE_MIDI_E6

LV_EVE_EVE_MIDI_F6

LV_EVE_EVE_MIDI_F_6

LV_EVE_EVE_MIDI_G6

LV_EVE_EVE_MIDI_G_6

LV_EVE_EVE_MIDI_A6

LV_EVE_EVE_MIDI_A_6

LV_EVE_EVE_MIDI_B6

LV_EVE_EVE_MIDI_C7

LV_EVE_EVE_MIDI_C_7

LV_EVE_EVE_MIDI_D7

LV_EVE_EVE_MIDI_D_7

LV_EVE_EVE_MIDI_E7

LV_EVE_EVE_MIDI_F7

LV_EVE_EVE_MIDI_F_7

LV_EVE_EVE_MIDI_G7

LV_EVE_EVE_MIDI_G_7

LV_EVE_EVE_MIDI_A7

LV_EVE_EVE_MIDI_A_7

LV_EVE_EVE_MIDI_B7

LV_EVE_EVE_MIDI_C8

LV_EVE_EVE_GPIO0

LV_EVE_EVE_GPIO1

LV_EVE_EVE_GPIO7

LV_EVE_EVE_DISPLAY_0

LV_EVE_EVE_DISPLAY_180

LV_EVE_CMD_APPEND

LV_EVE_CMD_BGCOLOR

LV_EVE_CMD_BUTTON

LV_EVE_CMD_CALIBRATE

LV_EVE_CMD_CLOCK

LV_EVE_CMD_COLDSTART

LV_EVE_CMD_DIAL

LV_EVE_CMD_DLSTART

LV_EVE_CMD_FGCOLOR

LV_EVE_CMD_GAUGE

LV_EVE_CMD_GETMATRIX

LV_EVE_CMD_GETPROPS

LV_EVE_CMD_GETPTR

LV_EVE_CMD_GRADCOLOR

LV_EVE_CMD_GRADIENT

LV_EVE_CMD_INFLATE

LV_EVE_CMD_INTERRUPT

LV_EVE_CMD_KEYS

LV_EVE_CMD_LOADIDENTITY

LV_EVE_CMD_LOADIMAGE

LV_EVE_CMD_LOGO

LV_EVE_CMD_MEDIAFIFO

LV_EVE_CMD_MEMCPY

LV_EVE_CMD_MEMCRC

LV_EVE_CMD_MEMSET

LV_EVE_CMD_MEMWRITE

LV_EVE_CMD_MEMZERO

LV_EVE_CMD_NUMBER

LV_EVE_CMD_PLAYVIDEO

LV_EVE_CMD_PROGRESS

LV_EVE_CMD_REGREAD

LV_EVE_CMD_ROMFONT

LV_EVE_CMD_ROTATE

LV_EVE_CMD_SCALE

LV_EVE_CMD_SCREENSAVER

LV_EVE_CMD_SCROLLBAR

LV_EVE_CMD_SETBASE

LV_EVE_CMD_SETBITMAP

LV_EVE_CMD_SETFONT

LV_EVE_CMD_SETFONT2

LV_EVE_CMD_SETMATRIX

LV_EVE_CMD_SETROTATE

LV_EVE_CMD_SETSCRATCH

LV_EVE_CMD_SKETCH

LV_EVE_CMD_SLIDER

LV_EVE_CMD_SNAPSHOT

LV_EVE_CMD_SNAPSHOT2

LV_EVE_CMD_SPINNER

LV_EVE_CMD_STOP

LV_EVE_CMD_SWAP

LV_EVE_CMD_TEXT

LV_EVE_CMD_TOGGLE

LV_EVE_CMD_TRACK

LV_EVE_CMD_TRANSLATE

LV_EVE_CMD_VIDEOFRAME

LV_EVE_CMD_VIDEOSTART

LV_EVE_REG_ANA_COMP

LV_EVE_REG_BIST_EN

LV_EVE_REG_CLOCK

LV_EVE_REG_CMDB_SPACE

LV_EVE_REG_CMDB_WRITE

LV_EVE_REG_CMD_DL

LV_EVE_REG_CMD_READ

LV_EVE_REG_CMD_WRITE

LV_EVE_REG_CPURESET

LV_EVE_REG_CSPREAD

LV_EVE_REG_CTOUCH_EXTENDED

LV_EVE_REG_CTOUCH_TOUCH0_XY

LV_EVE_REG_CTOUCH_TOUCH4_X

LV_EVE_REG_CTOUCH_TOUCH4_Y

LV_EVE_REG_CTOUCH_TOUCH1_XY

LV_EVE_REG_CTOUCH_TOUCH2_XY

LV_EVE_REG_CTOUCH_TOUCH3_XY

LV_EVE_REG_TOUCH_CONFIG

LV_EVE_REG_DATESTAMP

LV_EVE_REG_DITHER

LV_EVE_REG_DLSWAP

LV_EVE_REG_FRAMES

LV_EVE_REG_FREQUENCY

LV_EVE_REG_GPIO

LV_EVE_REG_GPIOX

LV_EVE_REG_GPIOX_DIR

LV_EVE_REG_GPIO_DIR

LV_EVE_REG_HCYCLE

LV_EVE_REG_HOFFSET

LV_EVE_REG_HSIZE

LV_EVE_REG_HSYNC0

LV_EVE_REG_HSYNC1

LV_EVE_REG_ID

LV_EVE_REG_INT_EN

LV_EVE_REG_INT_FLAGS

LV_EVE_REG_INT_MASK

LV_EVE_REG_MACRO_0

LV_EVE_REG_MACRO_1

LV_EVE_REG_MEDIAFIFO_READ

LV_EVE_REG_MEDIAFIFO_WRITE

LV_EVE_REG_OUTBITS

LV_EVE_REG_PCLK

LV_EVE_REG_PCLK_POL

LV_EVE_REG_PLAY

LV_EVE_REG_PLAYBACK_FORMAT

LV_EVE_REG_PLAYBACK_FREQ

LV_EVE_REG_PLAYBACK_LENGTH

LV_EVE_REG_PLAYBACK_LOOP

LV_EVE_REG_PLAYBACK_PLAY

LV_EVE_REG_PLAYBACK_READPTR

LV_EVE_REG_PLAYBACK_START

LV_EVE_REG_PWM_DUTY

LV_EVE_REG_PWM_HZ

LV_EVE_REG_RENDERMODE

LV_EVE_REG_ROTATE

LV_EVE_REG_SNAPFORMAT

LV_EVE_REG_SNAPSHOT

LV_EVE_REG_SNAPY

LV_EVE_REG_SOUND

LV_EVE_REG_SPI_WIDTH

LV_EVE_REG_SWIZZLE

LV_EVE_REG_TAG

LV_EVE_REG_TAG_X

LV_EVE_REG_TAG_Y

LV_EVE_REG_TAP_CRC

LV_EVE_REG_TAP_MASK

LV_EVE_REG_TOUCH_ADC_MODE

LV_EVE_REG_TOUCH_CHARGE

LV_EVE_REG_TOUCH_DIRECT_XY

LV_EVE_REG_TOUCH_DIRECT_Z1Z2

LV_EVE_REG_TOUCH_MODE

LV_EVE_REG_TOUCH_OVERSAMPLE

LV_EVE_REG_TOUCH_RAW_XY

LV_EVE_REG_TOUCH_RZ

LV_EVE_REG_TOUCH_RZTHRESH

LV_EVE_REG_TOUCH_SCREEN_XY

LV_EVE_REG_TOUCH_SETTLE

LV_EVE_REG_TOUCH_TAG

LV_EVE_REG_TOUCH_TAG1

LV_EVE_REG_TOUCH_TAG1_XY

LV_EVE_REG_TOUCH_TAG2

LV_EVE_REG_TOUCH_TAG2_XY

LV_EVE_REG_TOUCH_TAG3

LV_EVE_REG_TOUCH_TAG3_XY

LV_EVE_REG_TOUCH_TAG4

LV_EVE_REG_TOUCH_TAG4_XY

LV_EVE_REG_TOUCH_TAG_XY

LV_EVE_REG_TOUCH_TRANSFORM_A

LV_EVE_REG_TOUCH_TRANSFORM_B

LV_EVE_REG_TOUCH_TRANSFORM_C

LV_EVE_REG_TOUCH_TRANSFORM_D

LV_EVE_REG_TOUCH_TRANSFORM_E

LV_EVE_REG_TOUCH_TRANSFORM_F

LV_EVE_REG_TRACKER

LV_EVE_REG_TRACKER_1

LV_EVE_REG_TRACKER_2

LV_EVE_REG_TRACKER_3

LV_EVE_REG_TRACKER_4

LV_EVE_REG_TRIM

LV_EVE_REG_VCYCLE

LV_EVE_REG_VOFFSET

LV_EVE_REG_VOL_PB

LV_EVE_REG_VOL_SOUND

LV_EVE_REG_VSIZE

LV_EVE_REG_VSYNC0

LV_EVE_REG_VSYNC1

LV_EVE_EVE_GLFORMAT

LV_EVE_DL_BITMAP_EXT_FORMAT

LV_EVE_DL_BITMAP_SWIZZLE

LV_EVE_EVE_ASTC_4X4

LV_EVE_EVE_ASTC_5X4

LV_EVE_EVE_ASTC_5X5

LV_EVE_EVE_ASTC_6X5

LV_EVE_EVE_ASTC_6X6

LV_EVE_EVE_ASTC_8X5

LV_EVE_EVE_ASTC_8X6

LV_EVE_EVE_ASTC_8X8

LV_EVE_EVE_ASTC_10X5

LV_EVE_EVE_ASTC_10X6

LV_EVE_EVE_ASTC_10X8

LV_EVE_EVE_ASTC_10X10

LV_EVE_EVE_ASTC_12X10

LV_EVE_EVE_ASTC_12X12

LV_EVE_EVE_RAM_ERR_REPORT

LV_EVE_EVE_RAM_FLASH

LV_EVE_EVE_RAM_FLASH_POSTBLOB

LV_EVE_EVE_OPT_FLASH

LV_EVE_EVE_OPT_OVERLAY

LV_EVE_EVE_OPT_FORMAT

LV_EVE_EVE_OPT_FILL

LV_EVE_CMD_BITMAP_TRANSFORM

LV_EVE_CMD_SYNC

LV_EVE_CMD_FLASHERASE

LV_EVE_CMD_FLASHWRITE

LV_EVE_CMD_FLASHREAD

LV_EVE_CMD_FLASHUPDATE

LV_EVE_CMD_FLASHDETACH

LV_EVE_CMD_FLASHATTACH

LV_EVE_CMD_FLASHFAST

LV_EVE_CMD_FLASHSPIDESEL

LV_EVE_CMD_FLASHSPITX

LV_EVE_CMD_FLASHSPIRX

LV_EVE_CMD_FLASHSOURCE

LV_EVE_CMD_CLEARCACHE

LV_EVE_CMD_INFLATE2

LV_EVE_CMD_ROTATEAROUND

LV_EVE_CMD_RESETFONTS

LV_EVE_CMD_ANIMSTART

LV_EVE_CMD_ANIMSTOP

LV_EVE_CMD_ANIMXY

LV_EVE_CMD_ANIMDRAW

LV_EVE_CMD_GRADIENTA

LV_EVE_CMD_FILLWIDTH

LV_EVE_CMD_APPENDF

LV_EVE_CMD_ANIMFRAME

LV_EVE_CMD_VIDEOSTARTF

LV_EVE_REG_ADAPTIVE_FRAMERATE

LV_EVE_REG_PLAYBACK_PAUSE

LV_EVE_REG_FLASH_STATUS

LV_EVE_REG_FLASH_SIZE

LV_EVE_REG_PLAY_CONTROL

LV_EVE_REG_COPRO_PATCH_PTR

LV_EVE_CMD_ANIMFRAMERAM

LV_EVE_CMD_ANIMSTARTRAM

LV_EVE_CMD_APILEVEL

LV_EVE_CMD_CALIBRATESUB

LV_EVE_CMD_CALLLIST

LV_EVE_CMD_ENDLIST

LV_EVE_CMD_FLASHPROGRAM

LV_EVE_CMD_FONTCACHE

LV_EVE_CMD_FONTCACHEQUERY

LV_EVE_CMD_GETIMAGE

LV_EVE_CMD_HSF

LV_EVE_CMD_LINETIME

LV_EVE_CMD_NEWLIST

LV_EVE_CMD_PCLKFREQ

LV_EVE_CMD_RETURN

LV_EVE_CMD_RUNANIM

LV_EVE_CMD_TESTCARD

LV_EVE_CMD_WAIT

LV_EVE_REG_UNDERRUN

LV_EVE_REG_AH_HCYCLE_MAX

LV_EVE_REG_PCLK_FREQ

LV_EVE_REG_PCLK_2X

LV_EVE_REG_ANIM_ACTIVE

Functions

static inline uint32_t **LV_EVE_ALPHA_FUNC**(uint8_t func, uint8_t ref)

Set the alpha test function.

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_BITMAP_HANDLE**(uint8_t handle)

Set the bitmap handle.

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_BITMAP_LAYOUT**(uint8_t format, uint16_t linestride, uint16_t height)

Set the source bitmap memory format and layout for the current handle.

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_BITMAP_SIZE**(uint8_t filter, uint8_t wrapx, uint8_t wrapy, uint16_t width, uint16_t height)

Set the source bitmap memory format and layout for the current handle.

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_BITMAP_LAYOUT_H**(uint16_t linestride, uint16_t height)

Set the 2 most significant bits of the source bitmap memory format and layout for the current handle.

> **Nota**
>
> this is different to FTDIs implementation as this takes the original values as parameters and not only the upper bits

Parametri

- **linestride** -- 12-bit value specified to BITMAP_LAYOUT
- **height** -- 11-bit value specified to BITMAP_LAYOUT

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_BITMAP_SIZE_H**(uint16_t width, uint16_t height)

Set the 2 most significant bits of bitmaps dimension for the current handle.

> **Nota**
>
> this is different to FTDIs implementation as this takes the original values as parameters and not only the upper bits

Parametri

- **linestride** -- 11-bit value of bitmap width, the 2 most significant bits are used
- **height** -- 11-bit value of bitmap width, the 2 most significant bits are used

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_BITMAP_SOURCE**(uint32_t addr)

Set the source address of bitmap data in RAM_G or flash memory.

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_BITMAP_TRANSFORM_C**(uint32_t val)

Set the C coefficient of the bitmap transform matrix.

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_BITMAP_TRANSFORM_F**(uint32_t val)

Set the F coefficient of the bitmap transform matrix.

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_BLEND_FUNC**(uint8_t src, uint8_t dst)

Execute a sequence of commands at another location in the display list.

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_CALL**(uint16_t dest)

Execute a sequence of commands at another location in the display list.

> **Nota**
> valid range for dest is from zero to 2047

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_JUMP**(uint16_t dest)

Execute commands at another location in the display list.

> **Nota**
> valid range for dest is from zero to 2047

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_CELL**(uint8_t cell)

Set the bitmap cell number for the VERTEX2F command.

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_CLEAR**(uint8_t color, uint8_t stencil, uint8_t tag)

Clear buffers to preset values.

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_CLEAR_COLOR_A**(uint8_t alpha)

Set clear value for the alpha channel.

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_CLEAR_COLOR_RGB**(uint8_t red, uint8_t green, uint8_t blue)

Set clear values for red, green and blue channels.

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_CLEAR_STENCIL**(uint8_t val)

Set clear value for the stencil buffer.

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_CLEAR_TAG**(uint8_t val)

Set clear value for the tag buffer.

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_COLOR_A**(uint8_t alpha)

Set the current color alpha.

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_COLOR_MASK**(uint8_t red, uint8_t green, uint8_t blue, uint8_t alpha)

Enable or disable writing of color components.

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_COLOR_RGB**(uint8_t red, uint8_t green, uint8_t blue)

Set the current color red, green and blue.

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_LINE_WIDTH**(uint16_t width)

Set the width of lines to be drawn with primitive LINES in 1/16 pixel precision.

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_MACRO**(uint8_t macro)

Execute a single command from a macro register.

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_PALETTE_SOURCE**(uint32_t addr)

Set the base address of the palette.

> **Nota**
>
> 2-byte alignment is required if pixel format is PALETTE4444 or PALETTE565.

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_POINT_SIZE**(uint16_t size)

Set the radius of points to be drawn with primitive POINTS in 1/16 pixel precision.

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_SCISSOR_SIZE**(uint16_t width, uint16_t height)

Set the size of the scissor clip rectangle.

> **Nota**
>
> valid range for width and height is from zero to 2048

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_SCISSOR_XY**(uint16_t xc0, uint16_t yc0)

Set the top left corner of the scissor clip rectangle.

> **Nota**
>
> valid range for width and height is from zero to 2047

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_STENCIL_FUNC**(uint8_t func, uint8_t ref, uint8_t mask)

Set function and reference value for stencil testing.

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_STENCIL_MASK**(uint8_t mask)

Control the writing of individual bits in the stencil planes.

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_STENCIL_OP**(uint8_t sfail, uint8_t spass)

Set stencil test actions.

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_TAG**(uint8_t tagval)

Attach the tag value for the following graphics objects drawn on the screen.

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_TAG_MASK**(uint8_t mask)

Control the writing of the tag buffer.

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_VERTEX2F**(int16_t xc0, int16_t yc0)

Set coordinates for graphics primitves.

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_VERTEX2II**(uint16_t xc0, uint16_t yc0, uint8_t handle, uint8_t cell)

Set coordinates, bitmap-handle and cell-number for graphics primitves.

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_VERTEX_FORMAT**(uint8_t frac)

Set the precision of VERTEX2F coordinates.

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_VERTEX_TRANSLATE_X**(int32_t xco)

Set the vertex transformations X translation component.

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_VERTEX_TRANSLATE_Y**(int32_t yco)

Set the vertex transformations Y translation component.

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_BITMAP_EXT_FORMAT**(uint16_t format)

Set the extended format of the bitmap.

Ritorna
a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_BITMAP_SWIZZLE**(uint8_t red, uint8_t green, uint8_t blue, uint8_t alpha)

Set the source for the red, green, blue and alpha channels of a bitmap.

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_BITMAP_TRANSFORM_A**(uint8_t prc, uint32_t val)

Set the A coefficient of the bitmap transform matrix.

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_BITMAP_TRANSFORM_B**(uint8_t prc, uint32_t val)

Set the B coefficient of the bitmap transform matrix.

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_BITMAP_TRANSFORM_D**(uint8_t prc, uint32_t val)

Set the D coefficient of the bitmap transform matrix.

Ritorna

a 32 bit word for use with EVE_cmd_dl()

static inline uint32_t **LV_EVE_BITMAP_TRANSFORM_E**(uint8_t prc, uint32_t val)

Set the E coefficient of the bitmap transform matrix.

Ritorna

a 32 bit word for use with EVE_cmd_dl()

14.13.4 evdev

lv_evdev.h

Typedef

typedef void (*__lv_evdev_discovery_cb_t__)(lv_indev_t *indev, *lv_evdev_type_t* type, void *user_data)

Param indev

the indev created for the newly discovered evdev

Param type

the type of the evdev

Param user_data

a custom parameter

Enums

enum **lv_evdev_type_t**

Values:

enumerator **LV_EVDEV_TYPE_REL**

mice

enumerator **LV_EVDEV_TYPE_ABS**

touch screens, mousepads

enumerator **LV_EVDEV_TYPE_KEY**

keyboards, keypads, buttons

Functions

lv_indev_t ***lv_evdev_create**(*lv_indev_type_t* indev_type, const char *dev_path)

Create evdev input device.

Parametri

- **type** -- LV_INDEV_TYPE_POINTER or LV_INDEV_TYPE_KEYPAD
- **dev_path** -- device path, e.g., /dev/input/event0

Ritorna

pointer to input device or NULL if opening failed

lv_result_t **lv_evdev_discovery_start**(*lv_evdev_discovery_cb_t* cb, void *user_data)

Begin automatically creating evdev indevs for all new and existing evdev devices found in /dev/input/

Parametri

- **cb** -- function to call when a new evdev indev is discovered, or NULL
- **user_data** -- parameter to pass to the callback

Ritorna

the success or failure status. It will fail if it's already running or resources could not be initialized.

lv_result_t **lv_evdev_discovery_stop**(void)

Stop automatically creating evdev indevs. Safe to call from the discovery callback.

Ritorna

the success or failure status. It will fail if it's already running.

void **lv_evdev_set_swap_axes**(lv_indev_t *indev, bool swap_axes)

Set whether coordinates of pointer device should be swapped. Defaults to false.

Parametri

- **indev** -- evdev input device
- **swap_axes** -- whether to swap x and y axes

void **lv_evdev_set_calibration**(lv_indev_t *indev, int min_x, int min_y, int max_x, int max_y)

Configure a coordinate transformation for pointer devices. Applied after axis swap, if any. Defaults to apply no transformation.

Parametri

- **indev** -- evdev input device
- **min_x** -- pointer coordinate mapped to min x of display
- **min_y** -- pointer coordinate mapped to min y of display
- **max_x** -- pointer coordinate mapped to max x of display
- **max_y** -- pointer coordinate mapped to max y of display

void **lv_evdev_delete**(lv_indev_t *indev)

Remove evdev input device.

Parametri

indev -- evdev input device to close and free

lv_evdev_private.h

Functions

void **lv_evdev_deinit**(void)

14.13.5 glfw

lv_glfw_window.h

Functions

lv_glfw_window_t ***lv_glfw_window_create**(int32_t hor_res, int32_t ver_res, bool use_mouse_indev)

Create a GLFW window with no textures and initialize OpenGL

Parametri

- **hor_res** -- width in pixels of the window

- **ver_res** -- height in pixels of the window
- **use_mouse_indev** -- send pointer indev input to LVGL display textures

Ritorna
the new GLFW window handle

void **lv_glfw_window_delete**(lv_glfw_window_t *window)

Delete a GLFW window. If it is the last one, the process will exit

Parametri
window -- GLFW window to delete

void ***lv_glfw_window_get_glfw_window**(lv_glfw_window_t *window)

Get the GLFW window handle for an lv_glfw_window

Parametri
window -- GLFW window to return the handle of

Ritorna
the GLFW window handle

lv_glfw_texture_t ***lv_glfw_window_add_texture**(lv_glfw_window_t *window, unsigned int texture_id, int32_t w, int32_t h)

Add a texture to the GLFW window. It can be an LVGL display texture, or any OpenGL texture

Parametri
- **window** -- GLFW window
- **texture_id** -- OpenGL texture ID
- **w** -- width in pixels of the texture
- **h** -- height in pixels of the texture

Ritorna
the new texture handle

void **lv_glfw_texture_remove**(lv_glfw_texture_t *texture)

Remove a texture from its GLFW window and delete it

Parametri
texture -- handle of a GLFW window texture

void **lv_glfw_texture_set_x**(lv_glfw_texture_t *texture, int32_t x)

Set the x position of a texture within its GLFW window

Parametri
- **texture** -- handle of a GLFW window texture
- **x** -- new x position of the texture

void **lv_glfw_texture_set_y**(lv_glfw_texture_t *texture, int32_t y)

Set the y position of a texture within its GLFW window

Parametri
- **texture** -- handle of a GLFW window texture
- **y** -- new y position of the texture

void **lv_glfw_texture_set_opa**(lv_glfw_texture_t *texture, lv_opa_t opa)

Set the opacity of a texture in a GLFW window

Parametri
- **texture** -- handle of a GLFW window texture
- **opa** -- new opacity of the texture

lv_indev_t ***lv_glfw_texture_get_mouse_indev**(lv_glfw_texture_t *texture)

Get the mouse indev associated with a texture in a GLFW window, if it exists

> **Nota**
> there will only be an indev if the texture is based on an LVGL display texture and the window was created with `use_mouse_indev` as `true`

Parametri
texture -- handle of a GLFW window texture
Ritorna
the indev or NULL

lv_glfw_window_private.h

struct **_lv_glfw_window_t**

Public Members

GLFWwindow ***window**

int32_t **hor_res**

int32_t **ver_res**

lv_ll_t **textures**

lv_point_t **mouse_last_point**

lv_indev_state_t **mouse_last_state**

uint8_t **use_indev**

uint8_t **closing**

struct **_lv_glfw_texture_t**

Public Members

lv_glfw_window_t ***window**

unsigned int **texture_id**

lv_area_t **area**

lv_opa_t **opa**

lv_indev_t ***indev**

lv_point_t **indev_last_point**

lv_indev_state_t **indev_last_state**

lv_opengles_debug.h

Defines

GL_CALL(x)

Functions

void **GLClearError**(void)

void **GLLogCall**(const char *function, const char *file, int line)

lv_opengles_driver.h

Functions

void **lv_opengles_init**(void)

Initialize OpenGL

> **Nota**
>
> it is not necessary to call this if you use `lv_glfw_window_create`

void **lv_opengles_deinit**(void)

Deinitialize OpenGL

> **Nota**
>
> it is not necessary to call this if you use `lv_glfw_window_create`

void **lv_opengles_render_texture**(unsigned int texture, const *lv_area_t* *texture_area, lv_opa_t opa, int32_t disp_w, int32_t disp_h, const *lv_area_t* *texture_clip_area, bool flip)

Render a texture

Parametri

- **texture** -- OpenGL texture ID
- **texture_area** -- the area in the window to render the texture in
- **opa** -- opacity to blend the texture with existing contents
- **disp_w** -- width of the window/framebuffer being rendered to
- **disp_h** -- height of the window/framebuffer being rendered to

void **lv_opengles_render_fill**(*lv_color_t* color, const *lv_area_t* *area, lv_opa_t opa, int32_t disp_w, int32_t disp_h)

Render a fill

Parametri

- **color** -- the color of the fill
- **area** -- the area in the window to render the fill
- **opa** -- opacity to blend the fill with existing contents
- **disp_w** -- width of the window/framebuffer being rendered to
- **disp_h** -- height of the window/framebuffer being rendered to

void **lv_opengles_render_clear**(void)

Clear the window/display

void **lv_opengles_viewport**(int32_t x, int32_t y, int32_t w, int32_t h)

Set the OpenGL viewport

Parametri

- **x** -- x position of the viewport
- **y** -- y position of the viewport
- **w** -- width of the viewport
- **h** -- height of the viewport

lv_opengles_texture.h

Functions

lv_display_t ***lv_opengles_texture_create**(int32_t w, int32_t h)

Create a display that flushes to an OpenGL texture

Parametri

- **w** -- width in pixels of the texture
- **h** -- height in pixels of the texture

Ritorna

the new display

unsigned int **lv_opengles_texture_get_texture_id**(lv_display_t *disp)

Get the OpenGL texture ID of the display

Parametri

disp -- display

Ritorna

texture ID

lv_display_t ***lv_opengles_texture_get_from_texture_id**(unsigned int texture_id)

Get the display of an OpenGL texture if it is associated with one

Parametri

texture_id -- OpenGL texture ID

Ritorna

display or `NULL` if there no display with that texture ID

14.13.6 libinput

lv_libinput.h

Defines

LV_LIBINPUT_MAX_EVENTS

Enums

enum **lv_libinput_capability**

Values:

enumerator **LV_LIBINPUT_CAPABILITY_NONE**

enumerator **LV_LIBINPUT_CAPABILITY_KEYBOARD**

enumerator **LV_LIBINPUT_CAPABILITY_POINTER**

enumerator **LV_LIBINPUT_CAPABILITY_TOUCH**

Functions

lv_libinput_capability **lv_libinput_query_capability**(struct libinput_device *device)

Determine the capabilities of a specific libinput device.

Parametri

device -- the libinput device to query

Ritorna

the supported input capabilities

char ***lv_libinput_find_dev**(*lv_libinput_capability* capabilities, bool force_rescan)

Find connected input device with specific capabilities

Parametri

- **capabilities** -- required device capabilities
- **force_rescan** -- erase the device cache (if any) and rescan the file system for available devices

Ritorna

device node path (e.g. /dev/input/event0) for the first matching device or NULL if no device was found. The pointer is safe to use until the next forceful device search.

size_t **lv_libinput_find_devs**(*lv_libinput_capability* capabilities, char **found, size_t count, bool force_rescan)

Find connected input devices with specific capabilities

Parametri

- **capabilities** -- required device capabilities
- **devices** -- pre-allocated array to store the found device node paths (e.g. /dev/input/event0). The pointers are safe to use until the next forceful device search.
- **count** -- maximum number of devices to find (the devices array should be at least this long)
- **force_rescan** -- erase the device cache (if any) and rescan the file system for available devices

Ritorna

number of devices that were found

lv_indev_t ***lv_libinput_create**(*lv_indev_type_t* indev_type, const char *dev_path)

Create a new libinput input device

Parametri

- **type** -- LV_INDEV_TYPE_POINTER or LV_INDEV_TYPE_KEYPAD
- **dev_path** -- device path, e.g. /dev/input/event0

Ritorna

pointer to input device or NULL if opening failed

void **lv_libinput_delete**(lv_indev_t *indev)

Delete a libinput input device

Parametri

indev -- pointer to input device

lv_libinput_private.h

struct **_lv_libinput_event_t**

Public Members

lv_indev_state_t **pressed**

int **key_val**

lv_point_t **point**

struct **_lv_libinput_t**

Public Members

int **fd**

struct pollfd **fds**[1]

lv_libinput_event_t **points**[32]

lv_libinput_event_t **slots**[2]

lv_point_t **pointer_position**

bool **pointer_button_down**

int **start**

int **end**

lv_libinput_event_t **last_event**

bool **deinit**

pthread_mutex_t **event_lock**

pthread_t **worker_thread**

struct libinput ***libinput_context**

struct *libinput_device* ***libinput_device**

lv_xkb.h

lv_xkb_private.h

14.13.7 nuttx

lv_nuttx_cache.h

Functions

void **lv_nuttx_cache_init**(void)

void **lv_nuttx_cache_deinit**(void)

lv_nuttx_entry.h

Typedef

typedef struct *_lv_nuttx_ctx_t* **lv_nuttx_ctx_t**

Functions

void **lv_nuttx_dsc_init**(*lv_nuttx_dsc_t* *dsc)

Initialize the *lv_nuttx_dsc_t* structure with default values for the NuttX port of LVGL.

Parametri

dsc -- Pointer to the *lv_nuttx_dsc_t* structure to be initialized.

void **lv_nuttx_init**(const *lv_nuttx_dsc_t* *dsc, *lv_nuttx_result_t* *result)

Initialize the LVGL display driver for NuttX using the provided configuration information.

Parametri

- **dsc** -- Pointer to the *lv_nuttx_dsc_t* structure containing the configuration information for the display driver.
- **result** -- Pointer to the *lv_nuttx_result_t* structure containing display and input device handler.

void **lv_nuttx_deinit**(*lv_nuttx_result_t* *result)

Deinitialize the LVGL display driver for NuttX.

Parametri

result -- Pointer to the *lv_nuttx_result_t* structure containing display and input device handler.

void **lv_nuttx_init_custom**(const *lv_nuttx_dsc_t* *dsc, *lv_nuttx_result_t* *result)

Initialize the LVGL display driver for NuttX using the provided custom configuration information.

Parametri

- **dsc** -- Pointer to the *lv_nuttx_dsc_t* structure containing the custom configuration for the display driver.
- **result** -- Pointer to the *lv_nuttx_result_t* structure containing display and input device handler.

void **lv_nuttx_deinit_custom**(*lv_nuttx_result_t* *result)

Deinitialize the LVGL display driver for NuttX using the provided custom configuration information.

Parametri

result -- Pointer to the *lv_nuttx_result_t* structure containing display and input device handler.

void **lv_nuttx_run**(*lv_nuttx_result_t* *result)

Call `lv_timer_handler()` (LVGL's super loop) in an endless loop. If LV_USE_NUTTX_LIBUV is enabled an UV timer will be created, else `lv_timer_handler()` will be called in a loop with some sleep.

Parametri

result -- pointer to a variable initialized by `lv_nuttx_init()` or `lv_nuttx_init_custom()`

uint32_t **lv_nuttx_get_idle**(void)

Get the idle percentage of the system.

Ritorna

The idle percentage of the system.

struct **lv_nuttx_dsc_t**

Public Members

const char *`fb_path`

const char *`input_path`

const char *`utouch_path`

const char *`mouse_path`

struct `lv_nuttx_result_t`

Public Members

lv_display_t *`disp`

lv_indev_t *`indev`

lv_indev_t *`utouch_indev`

lv_indev_t *`mouse_indev`

struct `_lv_nuttx_ctx_t`

Public Members

void *`image_cache`

lv_nuttx_fbdev.h

Functions

lv_display_t *`lv_nuttx_fbdev_create`(void)

Create a new display with NuttX backend.

int `lv_nuttx_fbdev_set_file`(lv_display_t *disp, const char *file)

Initialize display with specified framebuffer device

Parametri

- `disp` -- pointer to display with NuttX backend
- `file` -- the name of framebuffer device

lv_nuttx_image_cache.h

Functions

void `lv_nuttx_image_cache_init`(bool use_independent_image_heap)

void `lv_nuttx_image_cache_deinit`(void)

lv_nuttx_lcd.h

Functions

lv_display_t *`lv_nuttx_lcd_create`(const char *dev_path)

lv_nuttx_libuv.h

Functions

void *`lv_nuttx_uv_init`(*lv_nuttx_uv_t* *uv_info)

Initialize the uv_loop using the provided configuration information.

Parametri

`uv_info` -- Pointer to the *lv_nuttx_uv_t* structure to be initialized.

void `lv_nuttx_uv_deinit`(void **data)

Deinitialize the uv_loop configuration for NuttX porting layer.

Parametri

`data` -- Pointer to user data.

struct `lv_nuttx_uv_t`

Public Members

void *`loop`

lv_display_t *`disp`

lv_indev_t *`indev`

lv_nuttx_mouse.h

Functions

lv_indev_t *`lv_nuttx_mouse_create`(const char *dev_path)

Initialize indev with specified input device.

Parametri

`dev_path` -- path of input device

lv_nuttx_profiler.h

Functions

void `lv_nuttx_profiler_init`(void)

lv_nuttx_touchscreen.h

Functions

lv_indev_t *`lv_nuttx_touchscreen_create`(const char *dev_path)

Initialize indev with specified input device.

Parametri

`dev_path` -- path of input device

14.13.8 qnx

lv_qnx.h

LVGL driver for the QNX Screen compositing window manager.

Functions

lv_display_t *`lv_qnx_window_create`(int32_t hor_res, int32_t ver_res)

Create a window to use as a display for LVGL.

Parametri

- `hor_res` -- The horizontal resolution (size) of the window
- `ver_res` -- The vertical resolution (size) of the window

Ritorna

A pointer to a new display object if successful, NULL otherwise

void `lv_qnx_window_set_title`(lv_display_t *disp, const char *title)

Set the title of the window identified by the given display.

Parametri

- `disp` -- The display object for the window
- `title` -- The new title to set

bool `lv_qnx_add_pointer_device`(lv_display_t *disp)

Create a pointer input device for the display. Only one pointer object is currently supported.

Parametri

`disp` -- The display object associated with the device

Ritorna

true if successful, false otherwise

bool `lv_qnx_add_keyboard_device`(lv_display_t *disp)

Create a keyboard input device for the display. Only one keyboard object is currently supported.

Parametri

`disp` -- The display object associated with the device

Ritorna

true if successful, false otherwise

int `lv_qnx_event_loop`(lv_display_t *disp)

Runs the event loop for the display. The function only returns in response to a close event.

Parametri

`disp` -- The display for the event loop

Ritorna

Exit code

14.13.9 sdl

lv_sdl_keyboard.h

Defines

KEYBOARD_BUFFER_SIZE

Functions

lv_indev_t *`lv_sdl_keyboard_create`(void)

lv_sdl_mouse.h

Functions

lv_indev_t *`lv_sdl_mouse_create`(void)

lv_sdl_mousewheel.h

Functions

lv_indev_t *`lv_sdl_mousewheel_create`(void)

lv_sdl_private.h

Functions

void `lv_sdl_keyboard_handler`(SDL_Event *event)

void `lv_sdl_mouse_handler`(SDL_Event *event)

void `lv_sdl_mousewheel_handler`(SDL_Event *event)

lv_display_t *`lv_sdl_get_disp_from_win_id`(uint32_t win_id)

lv_sdl_window.h

Defines

`LV_SDL_MOUSEWHEEL_MODE_ENCODER`

Modified by NXP in 2025

`LV_SDL_MOUSEWHEEL_MODE_CROWN`

Functions

lv_display_t *`lv_sdl_window_create`(int32_t hor_res, int32_t ver_res)

void `lv_sdl_window_set_resizeable`(lv_display_t *disp, bool value)

void `lv_sdl_window_set_zoom`(lv_display_t *disp, float zoom)

float `lv_sdl_window_get_zoom`(lv_display_t *disp)

void `lv_sdl_window_set_title`(lv_display_t *disp, const char *title)

void `lv_sdl_window_set_icon`(lv_display_t *disp, void *icon, int32_t width, int32_t height)

void *`lv_sdl_window_get_renderer`(lv_display_t *disp)

void `lv_sdl_quit`(void)

14.13.10 uefi

lv_uefi.h

lv_uefi_context.h

Functions

void **lv_uefi_init**(EFI_HANDLE image_handle, EFI_SYSTEM_TABLE *system_table)

Initialize the UEFI cache variables.

> **Remark**
>
> This has to be called before *lv_init()*.

Parametri

- **image_handle** -- The handle of the current image
- **system_table** -- Pointer to the system table

void **lv_uefi_platform_init**(void)

Initialize the LVGL UEFI backend.

> **Remark**
>
> This is a private API which is used for LVGL UEFI backend implementation. LVGL users shouldn't use that because the LVGL has already used it in lv_init.

void **lv_uefi_platform_deinit**(void)

Cleanup the LVGL UEFI backend.

> **Remark**
>
> This is a private API which is used for LVGL UEFI backend implementation. LVGL users shouldn't use that because the LVGL has already used it in lv_deinit.

lv_uefi_display.h

Functions

lv_display_t ***lv_uefi_display_create**(void *handle)

Create a LVGL display object.

Parametri

handle -- The handle on which an instance of the EFI_GRAPHICS_OUTPUT_PROTOCOL protocol is installed.

Ritorna

The created LVGL display object.

void ***lv_uefi_display_get_active**(void)

Try to find the active display handle.

> **Remark**
>
> The active display need interfaces for EFI_GRAPHICS_OUTPUT_PROTOCOL and EFI_EDID_ACTIVE_PROTOCOL

Ritorna

The handle or NULL if not found.

void ***lv_uefi_display_get_any**(void)

Try to find any display handle.

Ritorna

The handle or NULL if not found.

lv_uefi_edk2.h

lv_uefi_gnu_efi.h

lv_uefi_indev.h

Functions

lv_indev_t ***lv_uefi_simple_pointer_indev_create**(*lv_point_t* *display_res)

Create an indev object.

Parametri

display_res -- The resolution of the display in pixels, needed to scale the input. If NULL the resolution of the current default display will be used.

Ritorna

The created LVGL indev object.

bool **lv_uefi_simple_pointer_indev_add_handle**(lv_indev_t *indev, EFI_HANDLE handle)

Add an EFI_SIMPLE_POINTER_PROTOCOL interface to the indev.

Parametri

- **indev** -- Indev that was created with lv_uefi_simple_pointer_indev_create.
- **handle** -- The handle on which an instance of the EFI_SIMPLE_POINTER_PROTOCOL protocol is installed.

Ritorna

True if the interface was added.

void **lv_uefi_simple_pointer_indev_add_all**(lv_indev_t *indev)

Add all available EFI_SIMPLE_POINTER_PROTOCOL interfaces to the indev.

Parametri

indev -- Indev that was created with lv_uefi_simple_pointer_indev_create.

lv_indev_t ***lv_uefi_absolute_pointer_indev_create**(*lv_point_t* *display_res)

Create a LVGL indev object.

Parametri

display_res -- The resolution of the display in pixels, needed to scale the input.

Ritorna

The created LVGL indev object.

bool **lv_uefi_absolute_pointer_indev_add_handle**(lv_indev_t *indev, EFI_HANDLE handle)

Add an EFI_ABSOLUTE_POINTER_PROTOCOL interface to the indev.

Parametri

- **indev** -- Indev that was created with lv_uefi_absolute_pointer_indev_create.
- **handle** -- The handle on which an instance of the EFI_ABSOLUTE_POINTER_PROTOCOL protocol is installed.

Ritorna

True if the interface was added.

void **lv_uefi_absolute_pointer_indev_add_all**(lv_indev_t *indev)

Add all available EFI_ABSOLUTE_POINTER_PROTOCOL interfaces to the indev.

Parametri

indev -- Indev that was created with lv_uefi_absolute_pointer_indev_create.

lv_indev_t ***lv_uefi_simple_text_input_indev_create**(void)

Create an indev object.

Ritorna

The created LVGL indev object.

bool **lv_uefi_simple_text_input_indev_add_handle**(lv_indev_t *indev, EFI_HANDLE handle)

Add an EFI_SIMPLE_TEXT_INPUT_EX_PROTOCOL interface to the indev.

Parametri

- **indev** -- Indev that was created with lv_uefi_simple_text_input_indev_create.
- **handle** -- The handle on which an instance of the EFI_SIMPLE_TEXT_INPUT_EX_PROTOCOL protocol is installed.

Ritorna
True if the interface was added.

void **lv_uefi_simple_text_input_indev_add_all**(lv_indev_t *indev)

Add all available EFI_SIMPLE_TEXT_INPUT_EX_PROTOCOL interfaces to the indev.

Parametri
indev -- Indev that was created with lv_uefi_simple_text_input_indev_create.

lv_uefi_private.h

Functions

bool **lv_uefi_protocol_test**(EFI_HANDLE handle, EFI_GUID *protocol)

Test if a protocol is installed at a handle.

Parametri
- **handle** -- The handle on which the protocol might be installed.
- **protocol** -- The guid of the protocol.

Ritorna
TRUE if the protocol is installed, FALSE if not.

void ***lv_uefi_protocol_open**(EFI_HANDLE handle, EFI_GUID *protocol)

Open a protocol.

Parametri
- **handle** -- The handle on which the protocol is installed.
- **protocol** -- The guid of the protocol.

Ritorna
A pointer to the interface, NULL if the protocol couldn't be opened.

void **lv_uefi_protocol_close**(EFI_HANDLE handle, EFI_GUID *protocol)

Close a protocol.

Parametri
- **handle** -- The handle on which the protocol is installed.
- **protocol** -- The guid of the protocol.

size_t **lv_uefi_ucs2_to_ascii**(const CHAR16 *ucs2, char *ascii, size_t ascii_len)

Convert an UCS-2 string to an ASCII string. The string must contain only characters >= 0x20 and <= 0X7E.

Parametri
- **ucs2** -- The UCS-2 string.
- **ascii** -- The buffer to store the ASCII string.
- **ascii_len** -- The size of the buffer in ASCII characters.

Ritorna
The number of characters written to the buffer or 0 if there was an error.

size_t **lv_uefi_ascii_to_ucs2**(const char *ascii, CHAR16 *ucs2, size_t ucs2_len)

Convert an ASCII string to an UCS-2 string. The string must contain only characters >= 0x20 and <= 0X7E.

Parametri
- **ascii** -- The ASCII string.
- **ucs2** -- The buffer to store the UCS-2 string.
- **ucs2_len** -- The size of the buffer in UCS-2 characters.

Ritorna
The number of bytes written to the buffer or 0 if there was an error.

Variables

EFI_HANDLE **gLvEfiImageHandle**

Internal cache for the image handle (source: application entry point)

EFI_SYSTEM_TABLE ***gLvEfiST**

Internal cache for the system table (source: application entry point)

EFI_BOOT_SERVICES ***gLvEfiBS**

Internal cache for the boot services table (source: gLvEfiST)

EFI_RUNTIME_SERVICES ***gLvEfiRT**

Internal cache for the boot runtime service table (source: gLvEfiST)

lv_uefi_std_wrapper.h

14.13.11 wayland

lv_wayland.h

Functions

uint32_t **lv_wayland_timer_handler**(void)

Wrapper around lv_timer_handler

> **Nota**
>
> Must be called in the application run loop instead of the regular lv_timer_handler provided by LVGL

Ritorna

time till it needs to be run next (in ms)

int **lv_wayland_get_fd**(void)

Retrieves the file descriptor of the wayland socket

lv_wayland_private.h

Defines

LV_WAYLAND_XDG_SHELL

LV_WAYLAND_DEFAULT_CURSOR_NAME

LVGL_DRAW_BUFFER_DIV

DMG_CACHE_CAPACITY

TAG_LOCAL

TAG_BUFFER_DAMAGE

TITLE_BAR_HEIGHT

BORDER_SIZE

BUTTON_MARGIN

BUTTON_PADDING

BUTTON_SIZE

LV_WAYLAND_CYCLE_PERIOD

FIRST_DECORATION

LAST_DECORATION

NUM_DECORATIONS

Enums

enum **object_type**

Values:

enumerator **OBJECT_TITLEBAR**

enumerator **OBJECT_BUTTON_CLOSE**

enumerator **OBJECT_BUTTON_MAXIMIZE**

enumerator **OBJECT_BUTTON_MINIMIZE**

enumerator **OBJECT_BORDER_TOP**

enumerator **OBJECT_BORDER_BOTTOM**

enumerator **OBJECT_BORDER_LEFT**

enumerator **OBJECT_BORDER_RIGHT**

enumerator **OBJECT_WINDOW**

Functions

void **lv_wayland_init**(void)

void **lv_wayland_deinit**(void)

void **lv_wayland_wait_flush_cb**(lv_display_t *disp)

const struct wl_callback_listener ***lv_wayland_window_get_wl_surface_frame_listener**(void)

void **lv_wayland_window_draw**(struct *window* *window, uint32_t width, uint32_t height)

lv_result_t **lv_wayland_window_resize**(struct *window* *window, int width, int height)

void **lv_wayland_window_destroy**(struct *window* *window)

const struct xdg_surface_listener ***lv_wayland_xdg_shell_get_surface_listener**(void)

const struct xdg_toplevel_listener ***lv_wayland_xdg_shell_get_toplevel_listener**(void)

const struct xdg_wm_base_listener ***lv_wayland_xdg_shell_get_wm_base_listener**(void)

lv_result_t **lv_wayland_xdg_shell_set_maximized**(struct *window* *window, bool maximized)

lv_result_t **lv_wayland_xdg_shell_set_minimized**(struct *window* *window)

lv_result_t **lv_wayland_xdg_shell_set_fullscreen**(struct *window* *window, bool fullscreen)

lv_result_t **lv_wayland_xdg_shell_create_window**(struct *lv_wayland_context* *app, struct *window* *window, const char *title)

lv_result_t **lv_wayland_xdg_shell_destroy_window_toplevel**(struct *window* *window)

lv_result_t **lv_wayland_xdg_shell_destroy_window_surface**(struct *window* *window)

void **lv_wayland_xdg_shell_handle_pointer_event**(struct *lv_wayland_context* *app, uint32_t serial, uint32_t button, uint32_t state)

const char ***lv_wayland_xdg_shell_get_cursor_name**(const struct *lv_wayland_context* *app)

void **lv_wayland_xdg_shell_deinit**(void)

void **lv_wayland_shm_set_interface**(*shm_ctx_t* *context, struct wl_registry *registry, uint32_t name, const char *interface, uint32_t version)

struct *graphic_object* ***lv_wayland_shm_on_graphical_object_creation**(*shm_ctx_t* *context, struct *graphic_object* *obj)

void **lv_wayland_shm_on_graphical_object_destruction**(*shm_ctx_t* *context, struct *graphic_object* *obj)

lv_result_t **lv_wayland_shm_set_draw_buffers**(*shm_ctx_t* *context, lv_display_t *display)

lv_result_t **lv_wayland_shm_create_draw_buffers**(*shm_ctx_t* *context, struct *window* *window)

lv_result_t **lv_wayland_shm_resize_window**(*shm_ctx_t* *context, struct *window* *window, int32_t width, int32_t height)

lv_result_t **lv_wayland_shm_is_ready**(*shm_ctx_t* *context)

void **lv_wayland_shm_delete_draw_buffers**(*shm_ctx_t* *context, struct *window* *window)

void **lv_wayland_shm_initalize_context**(*shm_ctx_t* *context)

void **lv_wayland_shm_deinit**(*shm_ctx_t* *context)

void **lv_wayland_shm_flush_partial_mode**(lv_display_t *disp, const *lv_area_t* *area, unsigned char *color_p)

struct wl_cursor_theme ***lv_wayland_shm_load_cursor_theme**(*shm_ctx_t* *context)

void **lv_wayland_dmabuf_set_interface**(*dmabuf_ctx_t* *context, struct wl_registry *registry, uint32_t name, const char *interface, uint32_t version)

struct *graphic_object* ***lv_wayland_dmabuf_on_graphical_object_creation**(*dmabuf_ctx_t* *context, struct *graphic_object* *obj)

void **lv_wayland_dmabuf_on_graphical_object_destruction**(*dmabuf_ctx_t* *context, struct *graphic_object* *obj)

lv_result_t **lv_wayland_dmabuf_set_draw_buffers**(*dmabuf_ctx_t* *context, lv_display_t *display)

lv_result_t **lv_wayland_dmabuf_create_draw_buffers**(*dmabuf_ctx_t* *context, struct *window* *window)

lv_result_t **lv_wayland_dmabuf_resize_window**(*dmabuf_ctx_t* *context, struct *window* *window)

lv_result_t **lv_wayland_dmabuf_is_ready**(*dmabuf_ctx_t* *context)

void **lv_wayland_dmabuf_destroy_draw_buffers**(*dmabuf_ctx_t* *context, struct *window* *window)

void **lv_wayland_dmabuf_initalize_context**(*dmabuf_ctx_t* *context)

void **lv_wayland_dmabuf_deinit**(*dmabuf_ctx_t* *context)

void **lv_wayland_dmabuf_flush_full_mode**(lv_display_t *disp, const *lv_area_t* *area, unsigned char *color_p)

const struct *smm_events* ***lv_wayland_sme_get_events**(void)

const struct wl_seat_listener ***lv_wayland_seat_get_listener**(void)

const struct wl_keyboard_listener ***lv_wayland_keyboard_get_listener**(void)

const struct wl_pointer_listener ***lv_wayland_pointer_get_listener**(void)

const struct wl_touch_listener ***lv_wayland_touch_get_listener**(void)

void **lv_wayland_cache_add_area**(struct *window* *window, *smm_buffer_t* *buf, const *lv_area_t* *area)

void **lv_wayland_cache_clear**(struct *window* *window)

void **lv_wayland_cache_apply_areas**(struct *window* *window, void *dest, void *src, *smm_buffer_t* *src_buf)

void **lv_wayland_cache_purge**(struct *window* *window, *smm_buffer_t* *buf)

Variables

struct *lv_wayland_context* **lv_wl_ctx**

struct **input**

Public Members

uint32_t **x**

uint32_t **y**

lv_indev_state_t **left_button**

lv_indev_state_t **right_button**

lv_indev_state_t **wheel_button**

int16_t **wheel_diff**

struct *input* **pointer**

lv_key_t **key**

lv_indev_state_t **state**

struct *input* **keyboard**

lv_indev_touch_data_t **touches**[10]

uint8_t **touch_event_cnt**

uint8_t **primary_id**

struct **seat**

Public Members

struct *wl_touch* ***wl_touch**

struct *wl_pointer* ***wl_pointer**

struct *wl_keyboard* ***wl_keyboard**

struct xkb_keymap ***keymap**

struct xkb_state ***state**

struct *seat* **xkb**

struct **graphic_object**

Public Members

struct *window* ***window**

struct wl_surface ***surface**

struct wl_subsurface ***subsurface**

smm_buffer_t ***pending_buffer**

smm_group_t ***buffer_group**

struct *input* **input**

enum *object_type* **type**

int **width**

int **height**

struct **dmabuf_ctx_t**

Public Members

struct buffer ***buffers**

struct zwp_linux_dmabuf_v1 ***handler**

uint32_t **format**

struct **shm_ctx_t**

Public Members

lv_draw_buf_t ***lv_draw_buf**

struct wl_shm ***handler**

uint32_t **format**

struct **lv_wayland_context**

Public Members

struct wl_display ***display**

struct wl_registry ***registry**

struct wl_compositor ***compositor**

struct wl_subcompositor ***subcompositor**

struct *wl_seat* ***wl_seat**

struct wl_cursor_theme ***cursor_theme**

struct wl_surface ***cursor_surface**

shm_ctx_t **shm_ctx**

struct xdg_wm_base ***xdg_wm**

bool **opt_disable_decorations**

struct *xkb_context* ***xkb_context**

struct *seat* **seat**

struct *graphic_object* ***touch_obj**

struct *graphic_object* ***pointer_obj**

struct *graphic_object* ***keyboard_obj**

lv_ll_t **window_ll**

lv_timer_t ***cycle_timer**

bool **cursor_flush_pending**

struct pollfd **wayland_pfd**

struct **window**

Public Members

lv_display_t ***lv_disp**

lv_indev_t ***lv_indev_pointer**

lv_indev_t ***lv_indev_pointeraxis**

lv_indev_t ***lv_indev_touch**

lv_indev_t ***lv_indev_keyboard**

lv_wayland_display_close_f_t **close_cb**

struct *lv_wayland_context* ***wl_ctx**

struct *xdg_surface* ***xdg_surface**

struct *xdg_toplevel* ***xdg_toplevel**

uint32_t **wm_capabilities**

struct *graphic_object* ***body**

lv_area_t **cache**[(32)]

unsigned char **start**

unsigned char **end**

unsigned **size**

struct *window* **dmg_cache**

int **width**

int **height**

bool **resize_pending**

int **resize_width**

int **resize_height**

bool **flush_pending**

bool **shall_close**

bool **closed**

bool **maximized**

bool **fullscreen**

uint32_t **frame_counter**

lv_wayland_smm.h

Defines

SMM_FD_NAME

SMM_POOL_TAGS

SMM_BUFFER_TAGS

SMM_GROUP_TAGS

SMM_POOL_PROPERTIES(p)

SMM_BUFFER_PROPERTIES(b)

SMM_GROUP_PROPERTIES(g)

SMM_TAG(o, n, v)

Typedef

typedef void **smm_pool_t**

typedef void **smm_buffer_t**

typedef void **smm_group_t**

Functions

void **smm_init**(const struct *smm_events* *evs)

void **smm_setctx**(void *ctx)

void **smm_deinit**(void)

smm_group_t ***smm_create**(void)

void **smm_resize**(*smm_group_t* *grp, size_t sz)

void **smm_destroy**(*smm_group_t* *grp)

smm_buffer_t ***smm_acquire**(*smm_group_t* *grp)

void ***smm_map**(*smm_buffer_t* *buf)

void **smm_release**(*smm_buffer_t* *buf)

smm_buffer_t ***smm_latest**(*smm_group_t* *grp)

smm_buffer_t ***smm_next**(*smm_buffer_t* *buf)

struct **smm_events**

Public Members

void ***ctx**

bool (***new_pool**)(void *ctx, *smm_pool_t* *pool)

void (***expand_pool**)(void *ctx, *smm_pool_t* *pool)

void (***free_pool**)(void *ctx, *smm_pool_t* *pool)

bool (***new_buffer**)(void *ctx, *smm_buffer_t* *buf)

bool (*`init_buffer`)(void *ctx, *smm_buffer_t* *buf)

void (*`free_buffer`)(void *ctx, *smm_buffer_t* *buf)

struct `smm_pool_properties`

Public Members

void *`tag`[(1)]

size_t `size`

int `fd`

struct `smm_buffer_properties`

Public Members

void *`tag`[(2)]

smm_group_t *const `group`

smm_pool_t *const `pool`

size_t `offset`

struct `smm_group_properties`

Public Members

void *`tag`[(1)]

lv_wl_keyboard.h

Functions

lv_indev_t *`lv_wayland_keyboard_create`(void)

lv_indev_t *`lv_wayland_get_keyboard`(lv_display_t *display)

Get keyboard input device for given LVGL display

Parametri

`display` -- LVGL display

Ritorna

input device connected to keyboard, or NULL on error

lv_wl_pointer.h

Functions

lv_indev_t *`lv_wayland_pointer_create`(void)

lv_indev_t *`lv_wayland_get_pointer`(lv_display_t *disp)

Obtains the input device of the mouse pointer

> **Nota**
>
> It is used to create an input group on application start

Parametri

`disp` -- Reference to the LVGL display associated to the window

Ritorna

The input device

lv_wl_pointer_axis.h

Functions

lv_indev_t *`lv_wayland_pointer_axis_create`(void)

lv_indev_t *`lv_wayland_get_pointeraxis`(lv_display_t *display)

Obtains the input device of the encoder

> **Nota**
>
> It is used to create an input group on application start

Parametri

`display` -- Reference to the LVGL display associated to the window

Ritorna

The input device

lv_wl_touch.h

Functions

lv_indev_t *`lv_wayland_touch_create`(void)

lv_indev_t *`lv_wayland_get_touchscreen`(lv_display_t *display)

Get touchscreen input device for given LVGL display

Parametri

`display` -- LVGL display

Ritorna

input device connected to touchscreen, or NULL on error

lv_wl_window.h

Typedef

typedef bool (*`lv_wayland_display_close_f_t`)(lv_display_t *disp)

Functions

lv_display_t *`lv_wayland_window_create`(uint32_t hor_res, uint32_t ver_res, char *title, *lv_wayland_display_close_f_t* close_cb)

Creates a window

Parametri

- `hor_res` -- The width of the window in pixels
- `ver_res` -- The height of the window in pixels
- `title` -- The title of the window

- **close_cb** -- The callback that will be execute when the user closes the window

Ritorna
The LVGL display associated to the window

void **lv_wayland_window_close**(lv_display_t *disp)

Closes the window programmatically

Parametri
disp -- Reference to the LVGL display associated to the window

bool **lv_wayland_window_is_open**(lv_display_t *disp)

Check if the window is open

Parametri
disp -- Reference to the LVGL display associated to the window

Ritorna
true: The window is open

void **lv_wayland_window_set_fullscreen**(lv_display_t *disp, bool fullscreen)

Sets the fullscreen state of the window

Parametri
- **disp** -- Reference to the LVGL display associated to the window
- **fullscreen** -- If true the window enters fullscreen

void **lv_wayland_window_set_maximized**(lv_display_t *disp, bool maximize)

Sets the maximized state of the window

Parametri
- **disp** -- Reference to the LVGL display associated to the window
- **fullscreen** -- If true the window is maximized

14.13.12 windows

lv_windows_context.h

Defines

CREATE_WAITABLE_TIMER_MANUAL_RESET

CREATE_WAITABLE_TIMER_HIGH_RESOLUTION

Typedef

typedef struct *_lv_windows_pointer_context_t* **lv_windows_pointer_context_t**

typedef struct *_lv_windows_keypad_queue_item_t* **lv_windows_keypad_queue_item_t**

typedef struct *_lv_windows_keypad_context_t* **lv_windows_keypad_context_t**

typedef struct *_lv_windows_encoder_context_t* **lv_windows_encoder_context_t**

typedef struct *_lv_windows_window_context_t* **lv_windows_window_context_t**

typedef struct *_lv_windows_create_display_data_t* **lv_windows_create_display_data_t**

Functions

void **lv_windows_platform_init**(void)

Initialize the LVGL Windows backend.

> **Remark**
>
> This is a private API which is used for LVGL Windows backend implementation. LVGL users shouldn't use that because the LVGL has already used it in lv_init.

lv_windows_window_context_t ***lv_windows_get_window_context**(HWND window_handle)

Get the window context from specific LVGL display window.

> **Remark**
>
> This is a private API which is used for LVGL Windows backend implementation. LVGL users shouldn't use that because the maintainer doesn't promise the application binary interface compatibility for this API.

Parametri

window_handle -- The window handle of specific LVGL display window.

Ritorna

The window context from specific LVGL display window.

struct **_lv_windows_pointer_context_t**

Public Members

lv_indev_state_t **state**

lv_point_t **point**

lv_indev_t ***indev**

struct **_lv_windows_keypad_queue_item_t**

Public Members

uint32_t **key**

lv_indev_state_t **state**

struct **_lv_windows_keypad_context_t**

Public Members

lv_ll_t **queue**

uint16_t **utf16_high_surrogate**

uint16_t **utf16_low_surrogate**

lv_indev_t ***indev**

struct **_lv_windows_encoder_context_t**

Public Members

lv_indev_state_t **state**

int16_t **enc_diff**

lv_indev_t ***indev**

struct **_lv_windows_window_context_t**

Public Members

lv_display_t ***display_device_object**

lv_timer_t ***display_timer_object**

int32_t **window_dpi**

int32_t **zoom_level**

bool **allow_dpi_override**

bool **simulator_mode**

bool **display_resolution_changed**

lv_point_t **requested_display_resolution**

HDC **display_framebuffer_context_handle**

uint32_t ***display_framebuffer_base**

size_t **display_framebuffer_size**

lv_windows_pointer_context_t **pointer**

lv_windows_keypad_context_t **keypad**

lv_windows_encoder_context_t **encoder**

struct **_lv_windows_create_display_data_t**

Public Members

const wchar_t ***title**

int32_t **hor_res**

int32_t **ver_res**

int32_t **zoom_level**

bool **allow_dpi_override**

bool **simulator_mode**

HANDLE **mutex**

lv_display_t ***display**

lv_windows_display.h

Defines

LV_WINDOWS_ZOOM_BASE_LEVEL

USER_DEFAULT_SCREEN_DPI

Functions

lv_display_t ***lv_windows_create_display**(const wchar_t *title, int32_t hor_res, int32_t ver_res, int32_t zoom_level, bool allow_dpi_override, bool simulator_mode)

Create a LVGL display object.

Parametri

- **title** -- The window title of LVGL display.
- **hor_res** -- The horizontal resolution value of LVGL display.
- **ver_res** -- The vertical resolution value of LVGL display.
- **zoom_level** -- The zoom level value. Base value is 100 a.k.a 100%.
- **allow_dpi_override** -- Allow DPI override if true, or follow the Windows DPI scaling setting dynamically.
- **simulator_mode** -- Create simulator mode display if true, or create application mode display.

Ritorna

The created LVGL display object.

HWND **lv_windows_get_display_window_handle**(lv_display_t *display)

Get the window handle from specific LVGL display object.

Parametri

display -- The specific LVGL display object.

Ritorna

The window handle from specific LVGL display object.

int32_t **lv_windows_zoom_to_logical**(int32_t physical, int32_t zoom_level)

Get logical pixel value from physical pixel value taken account with zoom level.

> **Remark**
>
> It uses the same calculation style as Windows OS implementation. It will be useful for integrate LVGL Windows backend to other Windows applications.

Parametri

- **physical** -- The physical pixel value taken account with zoom level.
- **zoom_level** -- The zoom level value. Base value is 100 a.k.a 100%.

Ritorna

The logical pixel value.

int32_t **lv_windows_zoom_to_physical**(int32_t logical, int32_t zoom_level)

Get physical pixel value taken account with zoom level from logical pixel value.

> **Remark**
>
> It uses the same calculation style as Windows OS implementation. It will be useful for integrate LVGL Windows backend to other Windows applications.

Parametri

- **logical** -- The logical pixel value.
- **zoom_level** -- The zoom level value. Base value is 100 a.k.a 100%.

Ritorna

The physical pixel value taken account with zoom level.

int32_t **lv_windows_dpi_to_logical**(int32_t physical, int32_t dpi)

Get logical pixel value from physical pixel value taken account with DPI scaling.

> **Remark**
>
> It uses the same calculation style as Windows OS implementation. It will be useful for integrate LVGL Windows backend to other Windows applications.

Parametri

- **physical** -- The physical pixel value taken account with DPI scaling.
- **dpi** -- The DPI scaling value. Base value is USER_DEFAULT_SCREEN_DPI.

Ritorna

The logical pixel value.

int32_t **lv_windows_dpi_to_physical**(int32_t logical, int32_t dpi)

Get physical pixel value taken account with DPI scaling from logical pixel value.

> **Remark**
>
> It uses the same calculation style as Windows OS implementation. It will be useful for integrate LVGL Windows backend to other Windows applications.

Parametri

- **logical** -- The logical pixel value.
- **dpi** -- The DPI scaling value. Base value is USER_DEFAULT_SCREEN_DPI.

Ritorna

The physical pixel value taken account with DPI scaling.

lv_windows_input.h

Functions

HWND **lv_windows_get_indev_window_handle**(lv_indev_t *indev)

Get the window handle from specific LVGL input device object.

Parametri

indev -- The specific LVGL input device object.

Ritorna

The window handle from specific LVGL input device object.

lv_indev_t ***lv_windows_acquire_pointer_indev**(lv_display_t *display)

Open a LVGL pointer input device object for the specific LVGL display object, or create it if the LVGL pointer input device object is not created or removed before.

Parametri

display -- The specific LVGL display object.

Ritorna

The LVGL pointer input device object for the specific LVGL display object.

lv_indev_t ***lv_windows_acquire_keypad_indev**(lv_display_t *display)

Open a LVGL keypad input device object for the specific LVGL display object, or create it if the LVGL keypad input device object is not created or removed before.

Parametri

display -- The specific LVGL display object.

Ritorna

The LVGL keypad input device object for the specific LVGL display object.

lv_indev_t ***lv_windows_acquire_encoder_indev**(lv_display_t *display)

Open a LVGL encoder input device object for the specific LVGL display object, or create it if the LVGL encoder input device object is not created or removed before.

Parametri

display -- The specific LVGL display object.

Ritorna

The LVGL encoder input device object for the specific LVGL display object.

lv_windows_input_private.h

14.13.13 x11

lv_x11.h

Typedef

typedef void (***lv_x11_close_cb**)(void *user_data)

optional window close callback function type

> **Vedi anche**
>
> lv_x11_window_set_close_cb

Functions

void **lv_x11_inputs_create**(lv_display_t *disp, *lv_image_dsc_t* const *mouse_img)

create and add keyboard, mouse and scrollwheel objects and connect them to x11 display.

This is a convenience method handling the typical input initialisation of an X11 window:

- create keyboard (lv_x11_keyboard_create)
- create mouse (with scrollwheel, lv_x11_mouse_create lv_x11_mousewheel_create)

Parametri

- **disp** -- **[in]** the created X11 display object from *lv_x11_window_create*
- **mouse_img** -- **[in]** optional image description for the mouse cursor (NULL for no/invisible mouse cursor)

lv_display_t ***lv_x11_window_create**(char const *title, int32_t hor_res, int32_t ver_res)

create the X11 display

The minimal initialisation for initializing the X11 display driver with keyboard/mouse support:

```
lv_display_t* disp = lv_x11_window_create("My Window Title", window_
↪width, window_width);
lv_x11_inputs_create(disp, NULL);
```

or with mouse cursor icon:

```
lv_image_dsc_t mouse_symbol = {.....};
lv_display_t* disp = lv_x11_window_create("My Window Title", window_
↪width, window_width);
lv_x11_inputs_create(disp, &mouse_symbol);
```

Parametri

- **title** -- **[in]** title of the created X11 window
- **hor_res** -- **[in]** horizontal resolution (=width) of the X11 window
- **ver_res** -- **[in]** vertical resolution (=height) of the X11 window

Ritorna

pointer to the display object

struct **_x11_user_hdr_t**

#include <lv_x11.h> Header of private display driver user data - for internal use only

Public Members

struct _XDisplay ***display**

X11 display object

struct _x11_inp_data ***inp_data**

input user data object

14.14 font

14.14.1 lv_binfont_loader.h

Functions

lv_font_t ***lv_binfont_create**(const char *path)

Loads a `lv_font_t` object from a binary font file

Parametri

path -- path to font file

Ritorna

pointer to font where to load

void **lv_binfont_destroy**(lv_font_t *font)

Frees the memory allocated by the *`lv_binfont_create()`* function

Parametri

font -- lv_font_t object created by the lv_binfont_create function

Variables

const lv_font_class_t **lv_binfont_font_class**

struct **lv_binfont_font_src_t**

Public Members

uint32_t **font_size**

Size of the font in pixels

const char ***path**

Path to font file

const void ***buffer**

Address of the font file in the memory

uint32_t **buffer_size**

Size of the font file buffer

14.14.2 lv_font.h

Defines

LV_FONT_DECLARE(font_name)

Enums

enum **lv_font_glyph_format_t**

The font format.

Values:

enumerator **LV_FONT_GLYPH_FORMAT_NONE**

Maybe not visible Legacy simple formats with no byte padding at end of the lines

enumerator **LV_FONT_GLYPH_FORMAT_A1**

1 bit per pixel

enumerator **LV_FONT_GLYPH_FORMAT_A2**

2 bit per pixel

enumerator **LV_FONT_GLYPH_FORMAT_A3**

3 bit per pixel

enumerator **LV_FONT_GLYPH_FORMAT_A4**

4 bit per pixel

enumerator **LV_FONT_GLYPH_FORMAT_A8**

8 bit per pixel

enumerator **LV_FONT_GLYPH_FORMAT_IMAGE**

Image format Advanced formats

enumerator **LV_FONT_GLYPH_FORMAT_VECTOR**

Vectorial format

enumerator **LV_FONT_GLYPH_FORMAT_SVG**

SVG format

enumerator **LV_FONT_GLYPH_FORMAT_CUSTOM**

Custom format

enum **lv_font_subpx_t**

The bitmaps might be upscaled by 3 to achieve subpixel rendering.

Values:

enumerator **LV_FONT_SUBPX_NONE**

enumerator **LV_FONT_SUBPX_HOR**

enumerator **LV_FONT_SUBPX_VER**

enumerator **LV_FONT_SUBPX_BOTH**

enum **lv_font_kerning_t**

Adjust letter spacing for specific character pairs.

Values:

enumerator **LV_FONT_KERNING_NORMAL**

enumerator **LV_FONT_KERNING_NONE**

Functions

const void ***lv_font_get_glyph_bitmap**(*lv_font_glyph_dsc_t* *g_dsc, lv_draw_buf_t *draw_buf)

Return with the bitmap of a font. It always converts the normal fonts to A8 format in a draw_buf with LV_DRAW_BUF_ALIGN and LV_DRAW_BUF_STRIDE_ALIGN

> **Nota**
>
> You must call *lv_font_get_glyph_dsc()* to get `g_dsc` (*lv_font_glyph_dsc_t*) before you can call this function.

Parametri

- **g_dsc** -- the glyph descriptor including which font to use, which supply the glyph_index and the format.
- **draw_buf** -- a draw buffer that can be used to store the bitmap of the glyph.

Ritorna

pointer to the glyph's data. It can be a draw buffer for bitmap fonts or an image source for imgfonts.

const void *`lv_font_get_glyph_static_bitmap`(*lv_font_glyph_dsc_t* *g_dsc)

Return the bitmap as it is. It works only if the font stores the bitmap in a non-volitile memory.

Parametri

`g_dsc` -- the glyph descriptor including which font to use, which supply the glyph_index and the format.

Ritorna

the bitmap as it is

bool `lv_font_get_glyph_dsc`(const lv_font_t *font, *lv_font_glyph_dsc_t* *dsc_out, uint32_t letter, uint32_t letter_next)

Get the descriptor of a glyph

Parametri

- `font` -- pointer to font
- `dsc_out` -- store the result descriptor here
- `letter` -- a UNICODE letter code
- `letter_next` -- the next letter after `letter`. Used for kerning

Ritorna

true: descriptor is successfully loaded into `dsc_out`. false: the letter was not found, no data is loaded to `dsc_out`

void `lv_font_glyph_release_draw_data`(*lv_font_glyph_dsc_t* *g_dsc)

Release the bitmap of a font.

> **Nota**
>
> You must call *lv_font_get_glyph_dsc()* to get `g_dsc` (*lv_font_glyph_dsc_t*) before you can call this function.

Parametri

`g_dsc` -- the glyph descriptor including which font to use, which supply the glyph_index and the format.

uint16_t `lv_font_get_glyph_width`(const lv_font_t *font, uint32_t letter, uint32_t letter_next)

Get the width of a glyph with kerning

Parametri

- `font` -- pointer to a font
- `letter` -- a UNICODE letter
- `letter_next` -- the next letter after `letter`. Used for kerning

Ritorna

the width of the glyph

int32_t `lv_font_get_line_height`(const lv_font_t *font)

Get the line height of a font. All characters fit into this height

Parametri

`font` -- pointer to a font

Ritorna

the height of a font

void `lv_font_set_kerning`(lv_font_t *font, *lv_font_kerning_t* kerning)

Configure the use of kerning information stored in a font

Parametri

- `font` -- pointer to a font
- `kerning` -- `LV_FONT_KERNING_NORMAL` (default) or `LV_FONT_KERNING_NONE`

const lv_font_t *`lv_font_get_default`(void)

Get the default font, defined by LV_FONT_DEFAULT

Ritorna

return pointer to the default font

bool **lv_font_info_is_equal**(const lv_font_info_t *ft_info_1, const lv_font_info_t *ft_info_2)

Compare font information.

Parametri

- **ft_info_1** -- font information 1.
- **ft_info_2** -- font information 2.

Ritorna

return true if the fonts are equal.

bool **lv_font_has_static_bitmap**(const lv_font_t *font)

Checks if a font has a static rendering bitmap.

Parametri

font -- pointer to a font

Ritorna

return true if the font has a bitmap generated for static rendering.

Variables

const lv_font_t **lv_font_montserrat_8**

const lv_font_t **lv_font_montserrat_10**

const lv_font_t **lv_font_montserrat_12**

const lv_font_t **lv_font_montserrat_14**

const lv_font_t **lv_font_montserrat_16**

const lv_font_t **lv_font_montserrat_18**

const lv_font_t **lv_font_montserrat_20**

const lv_font_t **lv_font_montserrat_22**

const lv_font_t **lv_font_montserrat_24**

const lv_font_t **lv_font_montserrat_26**

const lv_font_t **lv_font_montserrat_28**

const lv_font_t **lv_font_montserrat_30**

const lv_font_t **lv_font_montserrat_32**

const lv_font_t **lv_font_montserrat_34**

const lv_font_t **lv_font_montserrat_36**

const lv_font_t **lv_font_montserrat_38**

const lv_font_t **lv_font_montserrat_40**

const lv_font_t **lv_font_montserrat_42**

const lv_font_t **lv_font_montserrat_44**

const lv_font_t **lv_font_montserrat_46**

const lv_font_t **lv_font_montserrat_48**

const lv_font_t **lv_font_montserrat_28_compressed**

const lv_font_t **lv_font_dejavu_16_persian_hebrew**

const lv_font_t **lv_font_source_han_sans_sc_14_cjk**

const lv_font_t **lv_font_source_han_sans_sc_16_cjk**

const lv_font_t **lv_font_unscii_8**

const lv_font_t **lv_font_unscii_16**

struct **lv_font_glyph_dsc_t**

#include <lv_font.h> Describes the properties of a glyph.

Public Members

const lv_font_t ***resolved_font**

Pointer to a font where the glyph was actually found after handling fallbacks

uint16_t **adv_w**

The glyph needs this space. Draw the next glyph after this width.

uint16_t **box_w**

Width of the glyph's bounding box

uint16_t **box_h**

Height of the glyph's bounding box

int16_t **ofs_x**

x offset of the bounding box

int16_t **ofs_y**

y offset of the bounding box

uint16_t **stride**

Bytes in each line. If 0 than there is no padding at the end of the line.

lv_font_glyph_format_t **format**

Font format of the glyph see *lv_font_glyph_format_t*

uint8_t **is_placeholder**

Glyph is missing. But placeholder will still be displayed

uint8_t **req_raw_bitmap**

0: Get bitmap should return an A8 or ARGB8888 image. 1: return the bitmap as it is (Maybe A1/2/4 or any proprietary formats).

int32_t **outline_stroke_width**

used with freetype vector fonts - width of the letter outline

uint32_t **index**

Unicode code point

const void ***src**

Pointer to the source data used by image fonts

union *lv_font_glyph_dsc_t* **gid**

The index of the glyph in the font file. Used by the font cache

lv_cache_entry_t ***entry**

The cache entry of the glyph draw data. Used by the font cache

struct **_lv_font_t**

#include <lv_font.h> Describe the properties of a font

Public Members

bool (***get_glyph_dsc**)(const lv_font_t*, *lv_font_glyph_dsc_t**, uint32_t letter, uint32_t letter_next)

Get a glyph's descriptor from a font

const void *(***get_glyph_bitmap**)(*lv_font_glyph_dsc_t**, lv_draw_buf_t*)

Get a glyph's bitmap from a font

void (***release_glyph**)(const lv_font_t*, *lv_font_glyph_dsc_t**)

Release a glyph

int32_t **line_height**

The real line height where any text fits

int32_t **base_line**

Base line measured from the bottom of the line_height

uint8_t **subpx**

An element of `lv_font_subpx_t`

uint8_t **kerning**

An element of *lv_font_kerning_t*

uint8_t **static_bitmap**

The font will be used as static bitmap

int8_t **underline_position**

Distance between the top of the underline and base line (< 0 means below the base line)

int8_t **underline_thickness**

Thickness of the underline

const void ***dsc**

Store implementation specific or run_time data or caching here

const lv_font_t ***fallback**

Fallback font for missing glyph. Resolved recursively

void ***user_data**

Custom user data for font.

struct **_lv_font_class_t**

Public Members

lv_font_t *(***create_cb**)(const lv_font_info_t *info, const void *src)

Font creation callback function

void (***delete_cb**)(lv_font_t *font)

Font deletion callback function

void *(***dup_src_cb**)(const void *src)

Font source duplication callback function

void (***free_src_cb**)(void *src)

Font source free callback function

struct **_lv_font_info_t**

Public Members

const char ***name**

Font name, used to distinguish different font resources

const lv_font_class_t ***class_p**

Font backend implementation

uint32_t **size**

Font size in pixel

uint32_t **render_mode**

Font rendering mode, see `lv_freetype_font_render_mode_t`

uint32_t **style**

Font style, see `lv_freetype_font_style_t`

lv_font_kerning_t **kerning**

Font kerning, see *`lv_font_kerning_t`*

14.14.3 lv_font_fmt_txt.h

Enums

enum **lv_font_fmt_txt_cmap_type_t**

Format of font character map.

Values:

enumerator **LV_FONT_FMT_TXT_CMAP_FORMAT0_FULL**

enumerator **LV_FONT_FMT_TXT_CMAP_SPARSE_FULL**

enumerator **LV_FONT_FMT_TXT_CMAP_FORMAT0_TINY**

enumerator **LV_FONT_FMT_TXT_CMAP_SPARSE_TINY**

enum **lv_font_fmt_txt_bitmap_format_t**

Bitmap formats

Values:

enumerator **LV_FONT_FMT_TXT_PLAIN**

enumerator **LV_FONT_FMT_TXT_COMPRESSED**

enumerator **LV_FONT_FMT_TXT_COMPRESSED_NO_PREFILTER**

Functions

const void ***lv_font_get_bitmap_fmt_txt**(*lv_font_glyph_dsc_t* *g_dsc, lv_draw_buf_t *draw_buf)

Used as `get_glyph_bitmap` callback in lvgl's native font format if the font is uncompressed.

Parametri

- **g_dsc** -- the glyph descriptor including which font to use, which supply the glyph_index and format.
- **draw_buf** -- a draw buffer that can be used to store the bitmap of the glyph, it's OK not to use it.

Ritorna

pointer to an A8 bitmap (not necessarily bitmap_out) or NULL if `unicode_letter` not found

bool **lv_font_get_glyph_dsc_fmt_txt**(const lv_font_t *font, *lv_font_glyph_dsc_t* *dsc_out, uint32_t unicode_letter, uint32_t unicode_letter_next)

Used as `get_glyph_dsc` callback in lvgl's native font format if the font is uncompressed.

Parametri

- **font** -- pointer to font
- **dsc_out** -- store the result descriptor here
- **unicode_letter** -- a UNICODE letter code
- **unicode_letter_next** -- the unicode letter succeeding the letter under test

Ritorna

true: descriptor is successfully loaded into `dsc_out`. false: the letter was not found, no data is loaded to `dsc_out`

Variables

const lv_font_class_t **lv_builtin_font_class**

struct **lv_font_fmt_txt_glyph_dsc_t**

#include <lv_font_fmt_txt.h> This describes a glyph.

Public Members

uint32_t **bitmap_index**

Start index of the bitmap. A font can be max 4 GB.

uint32_t **adv_w**

Draw the next glyph after this width. 28.4 format (real_value * 16 is stored).

uint16_t **box_w**

Width of the glyph's bounding box

uint16_t **box_h**

Height of the glyph's bounding box

int16_t **ofs_x**

x offset of the bounding box

int16_t **ofs_y**

y offset of the bounding box. Measured from the top of the line

struct **lv_font_fmt_txt_cmap_t**

#include <lv_font_fmt_txt.h> Map codepoints to a `glyph_dsc`s Several formats are supported to optimize memory usage See https://github.com/lvgl/lv_font_conv/blob/master/doc/font_spec.md

Public Members

uint32_t **range_start**

First Unicode character for this range

uint16_t **range_length**

Number of Unicode characters related to this range. Last Unicode character = range_start + range_length - 1

uint16_t **glyph_id_start**

First glyph ID (array index of `glyph_dsc`) for this range

const uint16_t ***unicode_list**

const void ***glyph_id_ofs_list**

if(type == LV_FONT_FMT_TXT_CMAP_FORMAT0_...) it's `uint8_t *` if(type == LV_FONT_FMT_TXT_CMAP_SPARSE_...) it's `uint16_t *`

uint16_t **list_length**

Length of `unicode_list` and/or `glyph_id_ofs_list`

lv_font_fmt_txt_cmap_type_t **type**

Type of this character map

struct **lv_font_fmt_txt_kern_pair_t**

#include <lv_font_fmt_txt.h> A simple mapping of kern values from pairs

Public Members

const void ***glyph_ids**

const int8_t ***values**

uint32_t **pair_cnt**

uint32_t **glyph_ids_size**

0: `glyph_ids` is stored as `uint8_t`; 1: as `uint16_t`

struct **lv_font_fmt_txt_kern_classes_t**

#include <lv_font_fmt_txt.h> More complex but more optimal class based kern value storage

Public Members

const int8_t ***class_pair_values**

left_class_cnt * right_class_cnt value

const uint8_t ***left_class_mapping**

Map the glyph_ids to classes: index -> glyph_id -> class_id

const uint8_t ***right_class_mapping**

Map the glyph_ids to classes: index -> glyph_id -> class_id

uint8_t **left_class_cnt**

uint8_t **right_class_cnt**

struct **lv_font_fmt_txt_dsc_t**

#include <lv_font_fmt_txt.h> Describe store for additional data for fonts

Public Members

const uint8_t *`glyph_bitmap`

The bitmaps of all glyphs

const *lv_font_fmt_txt_glyph_dsc_t* *`glyph_dsc`

Describe the glyphs

const *lv_font_fmt_txt_cmap_t* *`cmaps`

Map the glyphs to Unicode characters. Array of `lv_font_cmap_fmt_txt_t` variables

const void *`kern_dsc`

Store kerning values. Can be *`lv_font_fmt_txt_kern_pair_t`* `* or` lv_font_kern_classes_fmt_txt_t * `depending on` kern_classes`

uint16_t `kern_scale`

Scale kern values in 12.4 format

uint16_t `cmap_num`

Number of cmap tables

uint16_t `bpp`

Bit per pixel: 1, 2, 3, 4, 8

uint16_t `kern_classes`

Type of `kern_dsc`

uint16_t `bitmap_format`

storage format of the bitmap from *`lv_font_fmt_txt_bitmap_format_t`*

uint8_t `stride`

Bytes to which each line is padded. 0: means no align and padding 1: e.g. with bpp=4 lines are aligned to 1 byte, so there can be a 4 bits of padding 4, 8, 16, 32, 64: each line is padded to the given byte boundaries

struct `lv_builtin_font_src_t`

Public Members

const lv_font_t *`font_p`

Pointer to built-in font

uint32_t `size`

14.14.4 lv_font_fmt_txt_private.h

Enums

enum `lv_font_fmt_rle_state_t`

Values:

enumerator **RLE_STATE_SINGLE**

enumerator **RLE_STATE_REPEATED**

enumerator **RLE_STATE_COUNTER**

struct **lv_font_fmt_rle_t**

Public Members

uint32_t **rdp**

const uint8_t ***in**

uint8_t **bpp**

uint8_t **prev_v**

uint8_t **count**

lv_font_fmt_rle_state_t **state**

14.14.5 lv_symbol_def.h

Defines

LV_SYMBOL_BULLET

LV_SYMBOL_AUDIO

LV_SYMBOL_VIDEO

LV_SYMBOL_LIST

LV_SYMBOL_OK

LV_SYMBOL_CLOSE

LV_SYMBOL_POWER

LV_SYMBOL_SETTINGS

LV_SYMBOL_HOME

LV_SYMBOL_DOWNLOAD

LV_SYMBOL_DRIVE

LV_SYMBOL_REFRESH

LV_SYMBOL_MUTE

LV_SYMBOL_VOLUME_MID

LV_SYMBOL_VOLUME_MAX

LV_SYMBOL_IMAGE

LV_SYMBOL_TINT

LV_SYMBOL_PREV

LV_SYMBOL_PLAY

LV_SYMBOL_PAUSE

LV_SYMBOL_STOP

LV_SYMBOL_NEXT

LV_SYMBOL_EJECT

LV_SYMBOL_LEFT

LV_SYMBOL_RIGHT

LV_SYMBOL_PLUS

LV_SYMBOL_MINUS

LV_SYMBOL_EYE_OPEN

LV_SYMBOL_EYE_CLOSE

LV_SYMBOL_WARNING

LV_SYMBOL_SHUFFLE

LV_SYMBOL_UP

LV_SYMBOL_DOWN

LV_SYMBOL_LOOP

LV_SYMBOL_DIRECTORY

LV_SYMBOL_UPLOAD

LV_SYMBOL_CALL

LV_SYMBOL_CUT

LV_SYMBOL_COPY

LV_SYMBOL_SAVE

LV_SYMBOL_BARS

LV_SYMBOL_ENVELOPE

LV_SYMBOL_CHARGE

LV_SYMBOL_PASTE

LV_SYMBOL_BELL

LV_SYMBOL_KEYBOARD

LV_SYMBOL_GPS

LV_SYMBOL_FILE

LV_SYMBOL_WIFI

LV_SYMBOL_BATTERY_FULL

LV_SYMBOL_BATTERY_3

LV_SYMBOL_BATTERY_2

LV_SYMBOL_BATTERY_1

LV_SYMBOL_BATTERY_EMPTY

LV_SYMBOL_USB

LV_SYMBOL_BLUETOOTH

LV_SYMBOL_TRASH

LV_SYMBOL_EDIT

LV_SYMBOL_BACKSPACE

LV_SYMBOL_SD_CARD

LV_SYMBOL_NEW_LINE

LV_SYMBOL_DUMMY

Invalid symbol at (U+F8FF). If written before a string then `lv_img` will show it as a label

Enums

enum **_lv_str_symbol_id_t**

Values:

enumerator **LV_STR_SYMBOL_BULLET**

enumerator **LV_STR_SYMBOL_AUDIO**

enumerator **LV_STR_SYMBOL_VIDEO**

enumerator **LV_STR_SYMBOL_LIST**

enumerator **LV_STR_SYMBOL_OK**

enumerator **LV_STR_SYMBOL_CLOSE**

enumerator **LV_STR_SYMBOL_POWER**

enumerator **LV_STR_SYMBOL_SETTINGS**

enumerator **LV_STR_SYMBOL_HOME**

enumerator **LV_STR_SYMBOL_DOWNLOAD**

enumerator **LV_STR_SYMBOL_DRIVE**

enumerator **LV_STR_SYMBOL_REFRESH**

enumerator **LV_STR_SYMBOL_MUTE**

enumerator **LV_STR_SYMBOL_VOLUME_MID**

enumerator **LV_STR_SYMBOL_VOLUME_MAX**

enumerator **LV_STR_SYMBOL_IMAGE**

enumerator **LV_STR_SYMBOL_TINT**

enumerator **LV_STR_SYMBOL_PREV**

enumerator **LV_STR_SYMBOL_PLAY**

enumerator **LV_STR_SYMBOL_PAUSE**

enumerator **LV_STR_SYMBOL_STOP**

enumerator **LV_STR_SYMBOL_NEXT**

enumerator **LV_STR_SYMBOL_EJECT**

enumerator **LV_STR_SYMBOL_LEFT**

enumerator **LV_STR_SYMBOL_RIGHT**

enumerator **LV_STR_SYMBOL_PLUS**

enumerator **LV_STR_SYMBOL_MINUS**

enumerator **LV_STR_SYMBOL_EYE_OPEN**

enumerator **LV_STR_SYMBOL_EYE_CLOSE**

enumerator **LV_STR_SYMBOL_WARNING**

enumerator **LV_STR_SYMBOL_SHUFFLE**

enumerator **LV_STR_SYMBOL_UP**

enumerator **LV_STR_SYMBOL_DOWN**

enumerator **LV_STR_SYMBOL_LOOP**

enumerator **LV_STR_SYMBOL_DIRECTORY**

enumerator **LV_STR_SYMBOL_UPLOAD**

enumerator **LV_STR_SYMBOL_CALL**

enumerator **LV_STR_SYMBOL_CUT**

enumerator **LV_STR_SYMBOL_COPY**

enumerator **LV_STR_SYMBOL_SAVE**

enumerator **LV_STR_SYMBOL_BARS**

enumerator **LV_STR_SYMBOL_ENVELOPE**

enumerator **LV_STR_SYMBOL_CHARGE**

enumerator **LV_STR_SYMBOL_PASTE**

enumerator **LV_STR_SYMBOL_BELL**

enumerator **LV_STR_SYMBOL_KEYBOARD**

enumerator **LV_STR_SYMBOL_GPS**

enumerator **LV_STR_SYMBOL_FILE**

enumerator **LV_STR_SYMBOL_WIFI**

enumerator **LV_STR_SYMBOL_BATTERY_FULL**

enumerator **LV_STR_SYMBOL_BATTERY_3**

enumerator **LV_STR_SYMBOL_BATTERY_2**

enumerator **LV_STR_SYMBOL_BATTERY_1**

enumerator **LV_STR_SYMBOL_BATTERY_EMPTY**

enumerator **LV_STR_SYMBOL_USB**

enumerator **LV_STR_SYMBOL_BLUETOOTH**

enumerator **LV_STR_SYMBOL_TRASH**

enumerator **LV_STR_SYMBOL_EDIT**

enumerator **LV_STR_SYMBOL_BACKSPACE**

enumerator **LV_STR_SYMBOL_SD_CARD**

enumerator **LV_STR_SYMBOL_NEW_LINE**

enumerator **LV_STR_SYMBOL_DUMMY**

14.15 indev

14.15.1 lv_indev.h

Typedef

typedef void (*`lv_indev_read_cb_t`)(lv_indev_t *indev, *lv_indev_data_t* *data)

Enums

enum **lv_indev_type_t**

Possible input device types

Values:

enumerator **LV_INDEV_TYPE_NONE**

Uninitialized state

enumerator **LV_INDEV_TYPE_POINTER**

Touch pad, mouse, external button

enumerator **LV_INDEV_TYPE_KEYPAD**

Keypad or keyboard

enumerator **LV_INDEV_TYPE_BUTTON**

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

enumerator **LV_INDEV_TYPE_ENCODER**

Encoder with only Left, Right turn and a Button

enum **lv_indev_state_t**

States for input devices

Values:

enumerator **LV_INDEV_STATE_RELEASED**

enumerator **LV_INDEV_STATE_PRESSED**

enum **lv_indev_mode_t**

Values:

enumerator **LV_INDEV_MODE_NONE**

enumerator **LV_INDEV_MODE_TIMER**

enumerator **LV_INDEV_MODE_EVENT**

enum **lv_indev_gesture_type_t**

Values:

enumerator **LV_INDEV_GESTURE_NONE**

enumerator **LV_INDEV_GESTURE_PINCH**

enumerator **LV_INDEV_GESTURE_SWIPE**

enumerator **LV_INDEV_GESTURE_ROTATE**

enumerator **LV_INDEV_GESTURE_TWO_FINGERS_SWIPE**

enumerator **LV_INDEV_GESTURE_SCROLL**

enumerator **LV_INDEV_GESTURE_CNT**

Functions

lv_indev_t ***lv_indev_create**(void)

Create an indev

Ritorna

Pointer to the created indev or NULL when allocation failed

void **lv_indev_delete**(lv_indev_t *indev)

Remove the provided input device. Make sure not to use the provided input device afterwards anymore.

Parametri

indev -- pointer to delete

lv_indev_t ***lv_indev_get_next**(lv_indev_t *indev)

Get the next input device.

Parametri

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

Ritorna

the next input device or NULL if there are no more. Provide the first input device when the parameter is NULL

void **lv_indev_read**(lv_indev_t *indev)

Read data from an input device.

Parametri

indev -- pointer to an input device

void **lv_indev_read_timer_cb**(lv_timer_t *timer)

Called periodically to read the input devices

Parametri

timer -- pointer to a timer to read

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

Enable or disable one or all input devices (default enabled)

Parametri

- **indev** -- pointer to an input device or NULL to enable/disable all of them
- **enable** -- true to enable, false to disable

lv_indev_t ***lv_indev_active**(void)

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

Ritorna

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

void **lv_indev_set_type**(lv_indev_t *indev, *lv_indev_type_t* indev_type)

Set the type of an input device

Parametri

- **indev** -- pointer to an input device
- **indev_type** -- the type of the input device from `lv_indev_type_t` (`LV_INDEV_TYPE_...`)

void **lv_indev_set_read_cb**(lv_indev_t *indev, *lv_indev_read_cb_t* read_cb)

Set a callback function to read input device data to the indev

Parametri

- **indev** -- pointer to an input device
- **read_cb** -- pointer to callback function to read input device data

void **lv_indev_set_user_data**(lv_indev_t *indev, void *user_data)

Set user data to the indev

Parametri

- **indev** -- pointer to an input device
- **user_data** -- pointer to user data

void **lv_indev_set_driver_data**(lv_indev_t *indev, void *driver_data)

Set driver data to the indev

Parametri

- **indev** -- pointer to an input device
- **driver_data** -- pointer to driver data

void **lv_indev_set_display**(lv_indev_t *indev, struct *_lv_display_t* *disp)

Assign a display to the indev

Parametri

- **indev** -- pointer to an input device
- **disp** -- pointer to an display

void **lv_indev_set_long_press_time**(lv_indev_t *indev, uint16_t long_press_time)

Set long press time to indev

Parametri

- **indev** -- pointer to input device
- **long_press_time** -- time long press time in ms

void **lv_indev_set_long_press_repeat_time**(lv_indev_t *indev, uint16_t long_press_repeat_time)

Set long press repeat time to indev

Parametri

- **indev** -- pointer to input device
- **long_press_repeat_time** -- long press repeat time in ms

void **lv_indev_set_scroll_limit**(lv_indev_t *indev, uint8_t scroll_limit)

Set scroll limit to the input device

Parametri

- **indev** -- pointer to an input device
- **scroll_limit** -- the number of pixels to slide before actually drag the object

void **lv_indev_set_scroll_throw**(lv_indev_t *indev, uint8_t scroll_throw)

Set scroll throw slow-down to the indev. Greater value means faster slow-down

Parametri

- **indev** -- pointer to an input device
- **scroll_throw** -- the slow-down in [%]

lv_indev_type_t **lv_indev_get_type**(const lv_indev_t *indev)

Get the type of an input device

Parametri

indev -- pointer to an input device

Ritorna

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

lv_indev_read_cb_t **lv_indev_get_read_cb**(lv_indev_t *indev)

Get the callback function to read input device data to the indev

Parametri

indev -- pointer to an input device

Ritorna

Pointer to callback function to read input device data or NULL if indev is NULL

lv_indev_state_t **lv_indev_get_state**(const lv_indev_t *indev)

Get the indev state

Parametri

indev -- pointer to an input device

Ritorna

Indev state or LV_INDEV_STATE_RELEASED if indev is NULL

lv_group_t ***lv_indev_get_group**(const lv_indev_t *indev)

Get the indev assigned group

Parametri

indev -- pointer to an input device

Ritorna

Pointer to indev assigned group or NULL if indev is NULL

lv_display_t ***lv_indev_get_display**(const lv_indev_t *indev)

Get a pointer to the assigned display of the indev

Parametri

indev -- pointer to an input device

Ritorna

pointer to the assigned display or NULL if indev is NULL

void ***lv_indev_get_user_data**(const lv_indev_t *indev)

Get a pointer to the user data of the indev

Parametri

indev -- pointer to an input device

Ritorna

pointer to the user data or NULL if indev is NULL

void ***lv_indev_get_driver_data**(const lv_indev_t *indev)

Get a pointer to the driver data of the indev

Parametri

indev -- pointer to an input device

Ritorna

pointer to the driver data or NULL if indev is NULL

bool **lv_indev_get_press_moved**(const lv_indev_t *indev)

Get whether indev is moved while pressed

Parametri

indev -- pointer to an input device

Ritorna

true: indev is moved while pressed; false: indev is not moved while pressed

void **lv_indev_reset**(lv_indev_t *indev, lv_obj_t *obj)

Reset one or all input devices

Parametri

- **indev** -- pointer to an input device to reset or NULL to reset all of them

- **obj** -- pointer to an object which triggers the reset.

void **lv_indev_stop_processing**(lv_indev_t *indev)

Touch and key related events are sent to the input device first and to the widget after that. If this functions called in an indev event, the event won't be sent to the widget.

Parametri

indev -- pointer to an input device

void **lv_indev_reset_long_press**(lv_indev_t *indev)

Reset the long press state of an input device

Parametri

indev -- pointer to an input device

void **lv_indev_set_cursor**(lv_indev_t *indev, lv_obj_t *cur_obj)

Set a cursor for a pointer input device (for LV_INPUT_TYPE_POINTER and LV_INPUT_TYPE_BUTTON)

Parametri

- **indev** -- pointer to an input device
- **cur_obj** -- pointer to an object to be used as cursor

void **lv_indev_set_group**(lv_indev_t *indev, lv_group_t *group)

Set a destination group for a keypad input device (for LV_INDEV_TYPE_KEYPAD)

Parametri

- **indev** -- pointer to an input device
- **group** -- pointer to a group

void **lv_indev_set_button_points**(lv_indev_t *indev, const *lv_point_t* points[])

Set the an array of points for LV_INDEV_TYPE_BUTTON. These points will be assigned to the buttons to press a specific point on the screen

Parametri

- **indev** -- pointer to an input device
- **points** -- array of points

void **lv_indev_get_point**(const lv_indev_t *indev, *lv_point_t* *point)

Get the last point of an input device (for LV_INDEV_TYPE_POINTER and LV_INDEV_TYPE_BUTTON)

Parametri

- **indev** -- pointer to an input device
- **point** -- pointer to a point to store the result

lv_dir_t **lv_indev_get_gesture_dir**(const lv_indev_t *indev)

Get the current gesture direct

Parametri

indev -- pointer to an input device

Ritorna

current gesture direct

uint32_t **lv_indev_get_key**(const lv_indev_t *indev)

Get the last pressed key of an input device (for LV_INDEV_TYPE_KEYPAD)

Parametri

indev -- pointer to an input device

Ritorna

the last pressed key (0 on error)

uint8_t **lv_indev_get_short_click_streak**(const lv_indev_t *indev)

Get the counter for consecutive clicks within a short distance and time. The counter is updated before LV_EVENT_SHORT_CLICKED is fired.

Parametri

indev -- pointer to an input device

Ritorna

short click streak counter

lv_dir_t **lv_indev_get_scroll_dir**(const lv_indev_t *indev)

Check the current scroll direction of an input device (for LV_INDEV_TYPE_POINTER and LV_INDEV_TYPE_BUTTON)

Parametri

indev -- pointer to an input device

Ritorna

LV_DIR_NONE: no scrolling now LV_DIR_HOR/VER

lv_obj_t ***lv_indev_get_scroll_obj**(const lv_indev_t *indev)

Get the currently scrolled object (for LV_INDEV_TYPE_POINTER and LV_INDEV_TYPE_BUTTON)

Parametri

indev -- pointer to an input device

Ritorna

pointer to the currently scrolled object or NULL if no scrolling by this indev

void **lv_indev_get_vect**(const lv_indev_t *indev, *lv_point_t* *point)

Get the movement vector of an input device (for LV_INDEV_TYPE_POINTER and LV_INDEV_TYPE_BUTTON)

Parametri

- **indev** -- pointer to an input device
- **point** -- pointer to a point to store the types.pointer.vector

lv_obj_t ***lv_indev_get_cursor**(lv_indev_t *indev)

Get the cursor object of an input device (for LV_INDEV_TYPE_POINTER only)

Parametri

indev -- pointer to an input device

Ritorna

pointer to the cursor object

void **lv_indev_wait_release**(lv_indev_t *indev)

Do nothing until the next release

Parametri

indev -- pointer to an input device

lv_obj_t ***lv_indev_get_active_obj**(void)

Gets a pointer to the currently active object in the currently processed input device.

Ritorna

pointer to currently active object or NULL if no active object

lv_timer_t ***lv_indev_get_read_timer**(lv_indev_t *indev)

Get a pointer to the indev read timer to modify its parameters with `lv_timer_...` functions.

Parametri

indev -- pointer to an input device

Ritorna

pointer to the indev read refresher timer. (NULL on error)

void **lv_indev_set_mode**(lv_indev_t *indev, *lv_indev_mode_t* mode)

Set the input device's event model: event-driven mode or timer mode.

Parametri

- **indev** -- pointer to an input device
- **mode** -- the mode of input device

lv_indev_mode_t **lv_indev_get_mode**(lv_indev_t *indev)

Get the input device's running mode.

Parametri

indev -- pointer to an input device

Ritorna

the running mode for the specified input device.

lv_obj_t *`lv_indev_search_obj`(lv_obj_t *obj, *lv_point_t* *point)

Search the most top, clickable object by a point

Parametri

- **obj** -- pointer to a start object, typically the screen
- **point** -- pointer to a point for searching the most top child

Ritorna

pointer to the found object or NULL if there was no suitable object

void `lv_indev_add_event_cb`(lv_indev_t *indev, *lv_event_cb_t* event_cb, *lv_event_code_t* filter, void *user_data)

Add an event handler to the indev

Parametri

- **indev** -- pointer to an indev
- **event_cb** -- an event callback
- **filter** -- event code to react or `LV_EVENT_ALL`
- **user_data** -- optional user_data

uint32_t `lv_indev_get_event_count`(lv_indev_t *indev)

Get the number of event attached to an indev

Parametri

indev -- pointer to an indev

Ritorna

number of events

lv_event_dsc_t *`lv_indev_get_event_dsc`(lv_indev_t *indev, uint32_t index)

Get an event descriptor for an event

Parametri

- **indev** -- pointer to an indev
- **index** -- the index of the event

Ritorna

the event descriptor

bool `lv_indev_remove_event`(lv_indev_t *indev, uint32_t index)

Remove an event

Parametri

- **indev** -- pointer to an indev
- **index** -- the index of the event to remove

Ritorna

true: and event was removed; false: no event was removed

uint32_t `lv_indev_remove_event_cb_with_user_data`(lv_indev_t *indev, *lv_event_cb_t* event_cb, void *user_data)

Remove an event_cb with user_data

Parametri

- **indev** -- pointer to a indev
- **event_cb** -- the event_cb of the event to remove
- **user_data** -- user_data

Ritorna

the count of the event removed

lv_result_t `lv_indev_send_event`(lv_indev_t *indev, *lv_event_code_t* code, void *param)

Send an event to an indev

Parametri

- **indev** -- pointer to an indev
- **code** -- an event code. LV_EVENT_...
- **param** -- optional param

Ritorna

LV_RESULT_OK: indev wasn't deleted in the event.

struct **lv_indev_data_t**

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

Public Members

lv_indev_gesture_type_t **gesture_type**[*LV_INDEV_GESTURE_CNT*]

void ***gesture_data**[*LV_INDEV_GESTURE_CNT*]

lv_indev_state_t **state**

LV_INDEV_STATE_RELEASED or LV_INDEV_STATE_PRESSED

lv_point_t **point**

For LV_INDEV_TYPE_POINTER the currently pressed point

uint32_t **key**

For LV_INDEV_TYPE_KEYPAD the currently pressed key

uint32_t **btn_id**

For LV_INDEV_TYPE_BUTTON the currently pressed button

int16_t **enc_diff**

For LV_INDEV_TYPE_ENCODER number of steps since the previous read

uint32_t **timestamp**

Initialized to *lv_tick_get()*. Driver may provide more accurate timestamp for buffered events

bool **continue_reading**

If set to true, the read callback is invoked again, unless the device is in event-driven mode

14.15.2 lv_indev_gesture.h

Typedef

typedef struct *lv_indev_gesture_recognizer* **lv_indev_gesture_recognizer_t**

typedef struct *lv_indev_touch_data* **lv_indev_touch_data_t**

typedef struct *lv_indev_gesture* **lv_indev_gesture_t**

typedef struct *lv_indev_gesture_configuration* **lv_indev_gesture_configuration_t**

typedef void (***lv_recognizer_func_t**)(*lv_indev_gesture_recognizer_t**, *lv_indev_touch_data_t**, uint16_t)

Enums

enum **lv_indev_gesture_state_t**

Values:

enumerator **LV_INDEV_GESTURE_STATE_NONE**

enumerator **LV_INDEV_GESTURE_STATE_ONGOING**

enumerator **LV_INDEV_GESTURE_STATE_RECOGNIZED**

enumerator **LV_INDEV_GESTURE_STATE_ENDED**

enumerator **LV_INDEV_GESTURE_STATE_CANCELED**

Functions

void **lv_indev_gesture_detect_pinch**(*lv_indev_gesture_recognizer_t* *recognizer, *lv_indev_touch_data_t* *touches, uint16_t touch_cnt)

Pinch gesture recognizer function Will update the recognizer data

Parametri

- **recognizer** -- pointer to a gesture recognizer
- **touches** -- pointer to the first element of the collected touch events
- **touch_cnt** -- length of passed touch event array.

void **lv_indev_gesture_detect_rotation**(*lv_indev_gesture_recognizer_t* *recognizer, *lv_indev_touch_data_t* *touches, uint16_t touch_cnt)

Rotation gesture recognizer function Will update the recognizer data

Parametri

- **recognizer** -- pointer to a gesture recognizer
- **touches** -- pointer to the first element of the collected touch events
- **touch_cnt** -- length of passed touch event array.

void **lv_indev_gesture_detect_two_fingers_swipe**(*lv_indev_gesture_recognizer_t* *recognizer, *lv_indev_touch_data_t* *touches, uint16_t touch_cnt)

Two finger swipe gesture recognizer function Will update the recognizer data

Parametri

- **recognizer** -- pointer to a gesture recognizer
- **touches** -- pointer to the first element of the collected touch events
- **touch_cnt** -- length of passed touch event array.

void **lv_indev_set_pinch_up_threshold**(lv_indev_t *indev, float threshold)

Set the threshold for the pinch gesture scale up, when the scale factor of gesture reaches the threshold events get sent

Parametri

- **indev** -- pointer to the indev device containing the pinch recognizer
- **threshold** -- threshold for a pinch up gesture to be recognized

void **lv_indev_set_pinch_down_threshold**(lv_indev_t *indev, float threshold)

Set the threshold for the pinch gesture scale down, when the scale factor of gesture reaches the threshold events get sent

Parametri

- **indev** -- pointer to the indev device containing the pinch recognizer
- **threshold** -- threshold for a pinch down gesture to be recognized

void **lv_indev_set_rotation_rad_threshold**(lv_indev_t *indev, float threshold)

Set the rotation threshold in radian for the rotation gesture

Parametri

- **indev** -- pointer to the indev device containing the rotation recognizer
- **threshold** -- threshold in radian for a rotation gesture to be recognized

float **lv_event_get_pinch_scale**(lv_event_t *gesture_event)

Obtains the current scale of a pinch gesture

Parametri

gesture_event -- pointer to a gesture event

Ritorna

the scale of the current gesture

float **lv_event_get_rotation**(lv_event_t *gesture_event)

Obtains the current angle in radian of a rotation gesture

Parametri

gesture_event -- pointer to a gesture event

Ritorna

the rotation angle in radian of the current gesture

float **lv_event_get_two_fingers_swipe_distance**(lv_event_t *gesture_event)

Obtains the current distance in pixels of a two fingers swipe gesture, from the starting center

Parametri

gesture_event -- pointer to a gesture event

Ritorna

the distance from the center, in pixels, of the current gesture

lv_dir_t **lv_event_get_two_fingers_swipe_dir**(lv_event_t *gesture_event)

Obtains the current direction from the center of a two finger swipe

Parametri

gesture_event -- pointer to a gesture event

Ritorna

the rotation angle in radian of the current gesture

void **lv_indev_set_gesture_data**(*lv_indev_data_t* *data, *lv_indev_gesture_recognizer_t* *recognizer, *lv_indev_gesture_type_t* type)

Sets the state of the recognizer to a indev data structure, it is usually called from the indev read callback

Parametri

- **data** -- the indev data
- **recognizer** -- pointer to a gesture recognizer

void **lv_indev_get_gesture_center_point**(*lv_indev_gesture_recognizer_t* *recognizer, *lv_point_t* *point)

Obtains the center point of a gesture

Parametri

- **gesture_event** -- pointer to a gesture recognizer event
- **point** -- pointer to a point

lv_indev_gesture_state_t **lv_event_get_gesture_state**(lv_event_t *gesture_event, *lv_indev_gesture_type_t* type)

Obtains the current state of the gesture recognizer attached to an event

Parametri

gesture_event -- pointer to a gesture recognizer event

Ritorna

current state of the gesture recognizer

lv_indev_gesture_type_t **lv_event_get_gesture_type**(lv_event_t *gesture_event)

Obtains the current event type of the gesture recognizer attached to an event

Parametri

gesture_event -- pointer to a gesture recognizer event

Ritorna

current event type of the gesture recognizer

void **lv_indev_get_gesture_primary_point**(*lv_indev_gesture_recognizer_t* *recognizer, *lv_point_t* *point)

Obtains the coordinates of the current primary point

Parametri

- **recognizer** -- pointer to a gesture recognizer
- **point** -- pointer to a point

bool **lv_indev_recognizer_is_active**(*lv_indev_gesture_recognizer_t* *recognizer)

Allows to determine if there is an are ongoing gesture

Parametri

recognizer -- pointer to a gesture recognizer

Ritorna

false if there are no contact points, or the gesture has ended - true otherwise

void **lv_indev_gesture_recognizers_update**(lv_indev_t *indev, *lv_indev_touch_data_t* *touches, uint16_t touch_cnt)

Update the recognizers. It execute the recognizers functions and checks for LV_GESTURE_STATE_RECOGNIZED or LV_GESTURE_STATE_ENDED gestures. To be called in the indev read_cb.

Parametri

- **indev** -- pointer to the indev containing from which the reconizer need an update
- **touches** -- indev touch data array, containing the last touch data from indev since the last recognizers update
- **touch_cnt** -- number of indev touch data in touches

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

Set the *lv_indev_data_t* struct from the recognizer data. To be called in the indev read_cb.

struct **lv_indev_touch_data**

Public Members

lv_point_t **point**

lv_indev_state_t **state**

uint8_t **id**

uint32_t **timestamp**

struct **lv_indev_gesture_recognizer**

Public Members

lv_indev_gesture_type_t **type**

lv_indev_gesture_state_t **state**

lv_indev_gesture_t ***info**

float **scale**

float **rotation**

float **distance**

float **speed**

lv_dir_t **two_fingers_swipe_dir**

lv_indev_gesture_configuration_t ***config**

lv_recognizer_func_t **recog_fn**

14.15.3 lv_indev_gesture_private.h

Defines

LV_GESTURE_MAX_POINTS

Typedef

typedef struct *lv_indev_gesture_motion* **lv_indev_gesture_motion_t**

struct **lv_indev_gesture_motion**

Public Members

int8_t **finger**

lv_point_t **start_point**

lv_point_t **point**

lv_indev_state_t **state**

struct **lv_indev_gesture**

Public Members

lv_indev_gesture_motion_t **motions**[2]

lv_point_t **center**

float **scale**

float **p_scale**

float **scale_factors_x**[2]

float **scale_factors_y**[2]

float **delta_x**

float **delta_y**

float **p_delta_x**

float **p_delta_y**

float **rotation**

float **p_rotation**

uint8_t **finger_cnt**

struct **lv_indev_gesture_configuration**

Public Members

float **pinch_up_threshold**

float **pinch_down_threshold**

float **rotation_angle_rad_threshold**

14.15.4 lv_indev_private.h

Defines

LV_INDEV_VECT_HIST_SIZE

Functions

lv_obj_t ***lv_indev_find_scroll_obj**(lv_indev_t *indev)

Find a scrollable object based on the current scroll vector in the indev. In handles scroll propagation to the parent if needed, and scroll directions too.

Parametri

indev -- pointer to an indev

Ritorna

the found scrollable object or NULL if not found.

struct **_lv_indev_t**

Public Members

lv_indev_type_t **type**

Input device type

lv_indev_read_cb_t **read_cb**

Function pointer to read input device data.

lv_indev_state_t **state**

Current state of the input device.

lv_indev_state_t **prev_state**

Previous state of the input device.

lv_indev_mode_t **mode**

uint8_t **long_pr_sent**

uint8_t **reset_query**

uint8_t **enabled**

uint8_t **wait_until_release**

uint8_t **stop_processing_query**

uint32_t **timestamp**

Timestamp of last event

uint32_t **pr_timestamp**

Pressed time stamp

uint32_t **longpr_rep_timestamp**

Long press repeat time stamp

void ***driver_data**

void ***user_data**

Pointer to the assigned display

lv_display_t ***disp**

Timer to periodically read the input device

lv_timer_t ***read_timer**

Number of pixels to slide before actually drag the object

uint8_t **scroll_limit**

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

uint8_t **scroll_throw**

At least this difference should be between two points to evaluate as gesture

uint8_t **gesture_min_velocity**

At least this difference should be to send a gesture

uint8_t **gesture_limit**

Long press time in milliseconds

uint16_t **long_press_time**

Repeated trigger period in long press [ms]

uint16_t **long_press_repeat_time**

Rotary diff count will be multiplied by this value and divided by 256

int32_t **rotary_sensitivity**

lv_point_t **act_point**

Current point of input device.

lv_point_t **last_point**

Last point of input device.

lv_point_t **last_raw_point**

Last point read from read_cb.

lv_point_t **vect**

Difference between `act_point` and `last_point`.

lv_point_t **vect_hist**[8]

uint32_t **vect_hist_timestamp**[8]

uint8_t **vect_hist_index**

lv_point_t **scroll_sum**

lv_point_t **scroll_throw_vect**

lv_point_t **scroll_throw_vect_ori**

lv_obj_t ***act_obj**

lv_obj_t ***last_obj**

lv_obj_t ***scroll_obj**

lv_obj_t ***last_pressed**

lv_obj_t ***last_hovered**

lv_area_t **scroll_area**

lv_point_t **gesture_sum**

int32_t **diff**

uint8_t **short_click_streak**

lv_point_t **last_short_click_point**

uint32_t **last_short_click_timestamp**

uint8_t **scroll_dir**

uint8_t **gesture_dir**

uint8_t **gesture_sent**

uint8_t **press_moved**

uint8_t **pressed**

struct *_lv_indev_t* **pointer**

lv_indev_state_t **last_state**

uint32_t **last_key**

struct *_lv_indev_t* **keypad**

lv_obj_t ***cursor**

Cursor for LV_INPUT_TYPE_POINTER

lv_group_t ***group**

Keypad destination group

const *lv_point_t* ***btn_points**

Array points assigned to the button ()screen will be pressed here by the buttons

lv_event_list_t **event_list**

lv_anim_t ***scroll_throw_anim**

lv_indev_gesture_recognizer_t **recognizers**[*LV_INDEV_GESTURE_CNT*]

lv_indev_gesture_type_t **cur_gesture**

void ***gesture_data**[*LV_INDEV_GESTURE_CNT*]

lv_indev_gesture_type_t **gesture_type**[*LV_INDEV_GESTURE_CNT*]

14.15.5 lv_indev_scroll.h

Functions

void **lv_indev_scroll_handler**(lv_indev_t *indev)

Handle scrolling. Called by LVGL during input device processing

Parametri

indev -- pointer to an input device

void **lv_indev_scroll_throw_handler**(lv_indev_t *indev)

Handle throwing after scrolling. Called by LVGL during input device processing

Parametri

indev -- pointer to an input device

int32_t **lv_indev_scroll_throw_predict**(lv_indev_t *indev, *lv_dir_t* dir)

Predict where would a scroll throw end

Parametri

- **indev** -- pointer to an input device
- **dir** -- `LV_DIR_VER` or `LV_DIR_HOR`

Ritorna

the difference compared to the current position when the throw would be finished

void **lv_indev_scroll_get_snap_dist**(lv_obj_t *obj, *lv_point_t* *p)

Get the distance of the nearest snap point

Parametri

- **obj** -- the object on which snap points should be found
- **p** -- save the distance of the found snap point there

14.16 layouts

14.16.1 lv_layout.h

Typedef

typedef void (***lv_layout_update_cb_t**)(lv_obj_t*, void *user_data)

Enums

enum **lv_layout_t**

Values:

enumerator **LV_LAYOUT_NONE**

enumerator **LV_LAYOUT_FLEX**

enumerator **LV_LAYOUT_GRID**

enumerator **LV_LAYOUT_LAST**

Functions

uint32_t **lv_layout_register**(*lv_layout_update_cb_t* cb, void *user_data)

Register a new layout

Parametri

- **cb** -- the layout update callback
- **user_data** -- custom data that will be passed to cb

Ritorna

the ID of the new layout

14.16.2 lv_layout_private.h

Functions

void **lv_layout_init**(void)

void **lv_layout_deinit**(void)

void **lv_layout_apply**(lv_obj_t *obj)

Update the layout of a widget

Parametri

obj -- pointer to a widget

struct **lv_layout_dsc_t**

Public Members

lv_layout_update_cb_t **cb**

void ***user_data**

14.16.3 flex

lv_flex.h

Defines

LV_FLEX_COLUMN

LV_FLEX_WRAP

LV_FLEX_REVERSE

Enums

enum **lv_flex_align_t**

Values:

enumerator **LV_FLEX_ALIGN_START**

enumerator **LV_FLEX_ALIGN_END**

enumerator **LV_FLEX_ALIGN_CENTER**

enumerator **LV_FLEX_ALIGN_SPACE_EVENLY**

enumerator **LV_FLEX_ALIGN_SPACE_AROUND**

enumerator **LV_FLEX_ALIGN_SPACE_BETWEEN**

enum **lv_flex_flow_t**

Values:

enumerator **LV_FLEX_FLOW_ROW**

enumerator **LV_FLEX_FLOW_COLUMN**

enumerator **LV_FLEX_FLOW_ROW_WRAP**

enumerator **LV_FLEX_FLOW_ROW_REVERSE**

enumerator **LV_FLEX_FLOW_ROW_WRAP_REVERSE**

enumerator **LV_FLEX_FLOW_COLUMN_WRAP**

enumerator **LV_FLEX_FLOW_COLUMN_REVERSE**

enumerator **LV_FLEX_FLOW_COLUMN_WRAP_REVERSE**

Functions

void **lv_flex_init**(void)

Initialize a flex layout to default values

void **lv_obj_set_flex_flow**(lv_obj_t *obj, *lv_flex_flow_t* flow)

Set how the item should flow

Parametri

- **obj** -- pointer to an object. The parent must have flex layout else nothing will happen.
- **flow** -- an element of `lv_flex_flow_t`.

void **lv_obj_set_flex_align**(lv_obj_t *obj, *lv_flex_align_t* main_place, *lv_flex_align_t* cross_place, *lv_flex_align_t* track_cross_place)

Set how to place (where to align) the items and tracks

Parametri

- **obj** -- pointer to an object. The parent must have flex layout else nothing will happen.
- **main_place** -- where to place the items on main axis (in their track). Any value of `lv_flex_align_t`.
- **cross_place** -- where to place the item in their track on the cross axis. `LV_FLEX_ALIGN_START/END/CENTER`
- **track_cross_place** -- where to place the tracks in the cross direction. Any value of `lv_flex_align_t`.

void **lv_obj_set_flex_grow**(lv_obj_t *obj, uint8_t grow)

Sets the width or height (on main axis) to grow the object in order fill the free space

Parametri

- **obj** -- pointer to an object. The parent must have flex layout else nothing will happen.

- **grow** -- a value to set how much free space to take proportionally to other growing items.

14.16.4 grid

lv_grid.h

Defines

LV_GRID_FR(x)

Can be used track size to make the track fill the free space.

Parametri

- **x** -- how much space to take proportionally to other FR tracks

Ritorna

a special track size

LV_GRID_CONTENT

LV_GRID_TEMPLATE_LAST

Enums

enum **lv_grid_align_t**

Values:

enumerator **LV_GRID_ALIGN_START**

enumerator **LV_GRID_ALIGN_CENTER**

enumerator **LV_GRID_ALIGN_END**

enumerator **LV_GRID_ALIGN_STRETCH**

enumerator **LV_GRID_ALIGN_SPACE_EVENLY**

enumerator **LV_GRID_ALIGN_SPACE_AROUND**

enumerator **LV_GRID_ALIGN_SPACE_BETWEEN**

Functions

void **lv_grid_init**(void)

void **lv_obj_set_grid_dsc_array**(lv_obj_t *obj, const int32_t col_dsc[], const int32_t row_dsc[])

void **lv_obj_set_grid_align**(lv_obj_t *obj, *lv_grid_align_t* column_align, *lv_grid_align_t* row_align)

void **lv_obj_set_grid_cell**(lv_obj_t *obj, *lv_grid_align_t* column_align, int32_t col_pos, int32_t col_span, *lv_grid_align_t* row_align, int32_t row_pos, int32_t row_span)

Set the cell of an object. The object's parent needs to have grid layout, else nothing will happen

Parametri

- **obj** -- pointer to an object
- **column_align** -- the vertical alignment in the cell. `LV_GRID_START/END/CENTER/STRETCH`

- **col_pos** -- column ID
- **col_span** -- number of columns to take (>= 1)
- **row_align** -- the horizontal alignment in the cell. `LV_GRID_START/END/CENTER/STRETCH`
- **row_pos** -- row ID
- **row_span** -- number of rows to take (>= 1)

int32_t **lv_grid_fr**(uint8_t x)

Just a wrapper to `LV_GRID_FR` for bindings.

14.17 libs

14.17.1 barcode

lv_barcode.h

Enums

enum **lv_barcode_encoding_t**

Values:

enumerator **LV_BARCODE_ENCODING_CODE128_GS1**

Code 128 with GS1 encoding. Strips `[FCN1]` and spaces.

enumerator **LV_BARCODE_ENCODING_CODE128_RAW**

Code 128 with raw encoding.

Functions

lv_obj_t ***lv_barcode_create**(lv_obj_t *parent)

Create an empty barcode (an `lv_canvas`) object.

Parametri

parent -- point to an object where to create the barcode

Ritorna

pointer to the created barcode object

void **lv_barcode_set_dark_color**(lv_obj_t *obj, *lv_color_t* color)

Set the dark color of a barcode object

Parametri

- **obj** -- pointer to barcode object
- **color** -- dark color of the barcode

void **lv_barcode_set_light_color**(lv_obj_t *obj, *lv_color_t* color)

Set the light color of a barcode object

Parametri

- **obj** -- pointer to barcode object
- **color** -- light color of the barcode

void **lv_barcode_set_scale**(lv_obj_t *obj, uint16_t scale)

Set the scale of a barcode object

Parametri

- **obj** -- pointer to barcode object
- **scale** -- scale factor

void **lv_barcode_set_direction**(lv_obj_t *obj, *lv_dir_t* direction)

Set the direction of a barcode object

Parametri

- **obj** -- pointer to barcode object

- **direction** -- draw direction (LV_DIR_HOR or LB_DIR_VER)

void **lv_barcode_set_tiled**(lv_obj_t *obj, bool tiled)

Set the tiled mode of a barcode object

Parametri

- **obj** -- pointer to barcode object
- **tiled** -- true: tiled mode, false: normal mode (default)

void **lv_barcode_set_encoding**(lv_obj_t *obj, *lv_barcode_encoding_t* encoding)

Set the encoding of a barcode object

Parametri

- **obj** -- pointer to barcode object
- **encoding** -- encoding (default is LV_BARCODE_CODE128_GS1)

lv_result_t **lv_barcode_update**(lv_obj_t *obj, const char *data)

Set the data of a barcode object

Parametri

- **obj** -- pointer to barcode object
- **data** -- data to display

Ritorna

LV_RESULT_OK: if no error; LV_RESULT_INVALID: on error

lv_color_t **lv_barcode_get_dark_color**(lv_obj_t *obj)

Get the dark color of a barcode object

Parametri

obj -- pointer to barcode object

Ritorna

dark color of the barcode

lv_color_t **lv_barcode_get_light_color**(lv_obj_t *obj)

Get the light color of a barcode object

Parametri

obj -- pointer to barcode object

Ritorna

light color of the barcode

uint16_t **lv_barcode_get_scale**(lv_obj_t *obj)

Get the scale of a barcode object

Parametri

obj -- pointer to barcode object

Ritorna

scale factor

lv_barcode_encoding_t **lv_barcode_get_encoding**(const lv_obj_t *obj)

Get the encoding of a barcode object

Parametri

obj -- pointer to barcode object

Ritorna

encoding

Variables

const lv_obj_class_t **lv_barcode_class**

lv_barcode_private.h

struct **_lv_barcode_t**

Public Members

lv_canvas_t **canvas**

lv_color_t **dark_color**

lv_color_t **light_color**

uint16_t **scale**

lv_dir_t **direction**

bool **tiled**

lv_barcode_encoding_t **encoding**

14.17.2 bin_decoder

lv_bin_decoder.h

Functions

void **lv_bin_decoder_init**(void)

Initialize the binary image decoder module

lv_result_t **lv_bin_decoder_info**(lv_image_decoder_t *decoder, lv_image_decoder_dsc_t *dsc, *lv_image_header_t* *header)

Get info about a lvgl binary image

Parametri

- **decoder** -- the decoder where this function belongs
- **dsc** -- image descriptor containing the source and type of the image and other info.
- **header** -- store the image data here

Ritorna

LV_RESULT_OK: the info is successfully stored in `header`; LV_RESULT_INVALID: unknown format or other error.

lv_result_t **lv_bin_decoder_get_area**(lv_image_decoder_t *decoder, lv_image_decoder_dsc_t *dsc, const *lv_area_t* *full_area, *lv_area_t* *decoded_area)

lv_result_t **lv_bin_decoder_open**(lv_image_decoder_t *decoder, lv_image_decoder_dsc_t *dsc)

Open a lvgl binary image

Parametri

- **decoder** -- the decoder where this function belongs
- **dsc** -- pointer to decoder descriptor. `src`, `style` are already initialized in it.

Ritorna

LV_RESULT_OK: the info is successfully stored in `header`; LV_RESULT_INVALID: unknown format or other error.

void **lv_bin_decoder_close**(lv_image_decoder_t *decoder, lv_image_decoder_dsc_t *dsc)

Close the pending decoding. Free resources etc.

Parametri

- **decoder** -- pointer to the decoder the function associated with
- **dsc** -- pointer to decoder descriptor

14.17.3 bmp

lv_bmp.h

Functions

void **lv_bmp_init**(void)

void **lv_bmp_deinit**(void)

14.17.4 ffmpeg

lv_ffmpeg.h

Enums

enum **lv_ffmpeg_player_cmd_t**

Values:

enumerator **LV_FFMPEG_PLAYER_CMD_START**

enumerator **LV_FFMPEG_PLAYER_CMD_STOP**

enumerator **LV_FFMPEG_PLAYER_CMD_PAUSE**

enumerator **LV_FFMPEG_PLAYER_CMD_RESUME**

enumerator **LV_FFMPEG_PLAYER_CMD_LAST**

Functions

void **lv_ffmpeg_init**(void)

Register FFMPEG image decoder

void **lv_ffmpeg_deinit**(void)

De-initialize FFMPEG image decoder

int **lv_ffmpeg_get_frame_num**(const char *path)

Get the number of frames contained in the file

Parametri

path -- image or video file name

Ritorna

Number of frames, less than 0 means failed

lv_obj_t ***lv_ffmpeg_player_create**(lv_obj_t *parent)

Create ffmpeg_player object

Parametri

parent -- pointer to an object, it will be the parent of the new player

Ritorna

pointer to the created ffmpeg_player

lv_result_t **lv_ffmpeg_player_set_src**(lv_obj_t *obj, const char *path)

Set the path of the file to be played.

Parametri

- **obj** -- pointer to a ffmpeg_player object
- **path** -- video file path

Ritorna

LV_RESULT_OK: no error; LV_RESULT_INVALID: can't get the info.

void **lv_ffmpeg_player_set_cmd**(lv_obj_t *obj, *lv_ffmpeg_player_cmd_t* cmd)

Set command control video player

Parametri

- **obj** -- pointer to a ffmpeg_player object
- **cmd** -- control commands

void **lv_ffmpeg_player_set_auto_restart**(lv_obj_t *obj, bool en)

Set the video to automatically replay

Parametri

- **obj** -- pointer to a ffmpeg_player object
- **en** -- true: enable the auto restart

Variables

const lv_obj_class_t **lv_ffmpeg_player_class**

lv_ffmpeg_private.h

struct **_lv_ffmpeg_player_t**

Public Members

lv_image_t **img**

lv_timer_t ***timer**

lv_image_dsc_t **imgdsc**

bool **auto_restart**

struct ffmpeg_context_s ***ffmpeg_ctx**

14.17.5 freetype

lv_freetype.h

Defines

LV_FREETYPE_F26DOT6_TO_INT(x)

LV_FREETYPE_F26DOT6_TO_FLOAT(x)

FT_FONT_STYLE_NORMAL

FT_FONT_STYLE_ITALIC

FT_FONT_STYLE_BOLD

Typedef

typedef *lv_freetype_font_style_t* **LV_FT_FONT_STYLE**

typedef void *`lv_freetype_outline_t`

typedef const char `lv_freetype_font_src_t`

Enums

enum `lv_freetype_font_style_t`

Values:

enumerator `LV_FREETYPE_FONT_STYLE_NORMAL`

enumerator `LV_FREETYPE_FONT_STYLE_ITALIC`

enumerator `LV_FREETYPE_FONT_STYLE_BOLD`

enum `lv_freetype_font_render_mode_t`

Values:

enumerator `LV_FREETYPE_FONT_RENDER_MODE_BITMAP`

enumerator `LV_FREETYPE_FONT_RENDER_MODE_OUTLINE`

enum `lv_freetype_outline_type_t`

Values:

enumerator `LV_FREETYPE_OUTLINE_END`

enumerator `LV_FREETYPE_OUTLINE_MOVE_TO`

enumerator `LV_FREETYPE_OUTLINE_LINE_TO`

enumerator `LV_FREETYPE_OUTLINE_CUBIC_TO`

enumerator `LV_FREETYPE_OUTLINE_CONIC_TO`

enumerator `LV_FREETYPE_OUTLINE_BORDER_START`

Functions

lv_result_t `lv_freetype_init`(uint32_t max_glyph_cnt)

Initialize the freetype library.

Ritorna

LV_RESULT_OK on success, otherwise LV_RESULT_INVALID.

void `lv_freetype_uninit`(void)

Uninitialize the freetype library

void **lv_freetype_init_font_info**(lv_font_info_t *font_info)

Initialize a font info structure.

Parametri

font_info -- font info structure to be initialized.

lv_font_t ***lv_freetype_font_create_with_info**(const lv_font_info_t *font_info)

Create a freetype font with a font info structure.

Parametri

font_info -- font info structure.

Ritorna

Created font, or NULL on failure.

lv_font_t ***lv_freetype_font_create**(const char *pathname, *lv_freetype_font_render_mode_t* render_mode, uint32_t size, *lv_freetype_font_style_t* style)

Create a freetype font.

Parametri

- **pathname** -- font file path.
- **render_mode** -- font render mode(see @lv_freetype_font_render_mode_t for details).
- **size** -- font size.
- **style** -- font style(see lv_freetype_font_style_t for details).

Ritorna

Created font, or NULL on failure.

void **lv_freetype_font_delete**(lv_font_t *font)

Delete a freetype font.

Parametri

font -- freetype font to be deleted.

void **lv_freetype_outline_add_event**(*lv_event_cb_t* event_cb, *lv_event_code_t* filter, void *user_data)

Register a callback function to generate outlines for FreeType fonts.

Parametri

- **cb** -- The callback function to be registered.
- **user_data** -- User data to be passed to the callback function.

Ritorna

The ID of the registered callback function, or a negative value on failure.

uint32_t **lv_freetype_outline_get_scale**(const lv_font_t *font)

Get the scale of a FreeType font.

Parametri

font -- The FreeType font to get the scale of.

Ritorna

The scale of the FreeType font.

bool **lv_freetype_is_outline_font**(const lv_font_t *font)

Check if the font is an outline font.

Parametri

font -- The FreeType font.

Ritorna

Is outline font on success, otherwise false.

Variables

const lv_font_class_t **lv_freetype_font_class**

lv_freetype_private.h

Defines

FT_ERROR_MSG(msg, error_code)

LV_FREETYPE_FONT_DSC_MAGIC_NUM

LV_FREETYPE_FONT_DSC_HAS_MAGIC_NUM(dsc)

LV_ASSERT_FREETYPE_FONT_DSC(dsc)

FT_INT_TO_F26DOT6(x)

FT_F26DOT6_TO_INT(x)

FT_INT_TO_F16DOT16(x)

FT_F16DOT16_TO_INT(x)

Typedef

typedef struct *_lv_freetype_cache_node_t* **lv_freetype_cache_node_t**

typedef struct *_lv_freetype_context_t* **lv_freetype_context_t**

typedef struct *_lv_freetype_font_dsc_t* **lv_freetype_font_dsc_t**

Functions

lv_freetype_context_t ***lv_freetype_get_context**(void)

Get the FreeType context.

Ritorna

A pointer to the FreeType context used by LittlevGL.

void **lv_freetype_italic_transform**(FT_Face face)

int32_t **lv_freetype_italic_transform_on_pos**(*lv_point_t* point)

lv_cache_t ***lv_freetype_create_glyph_cache**(uint32_t cache_size)

void **lv_freetype_set_cbs_glyph**(*lv_freetype_font_dsc_t* *dsc)

lv_cache_t ***lv_freetype_create_draw_data_image**(uint32_t cache_size)

void **lv_freetype_set_cbs_image_font**(*lv_freetype_font_dsc_t* *dsc)

lv_cache_t ***lv_freetype_create_draw_data_outline**(uint32_t cache_size)

void **lv_freetype_set_cbs_outline_font**(*lv_freetype_font_dsc_t* *dsc)

struct **_lv_freetype_outline_vector_t**

Public Members

int32_t **x**

int32_t **y**

struct **lv_freetype_outline_sizes_t**

Public Members

int32_t **segments_size**

int32_t **data_size**

struct **_lv_freetype_outline_event_param_t**

Public Members

lv_freetype_outline_t **outline**

lv_freetype_outline_type_t **type**

lv_freetype_outline_vector_t **to**

lv_freetype_outline_vector_t **control1**

lv_freetype_outline_vector_t **control2**

lv_freetype_outline_sizes_t **sizes**

struct **_lv_freetype_cache_node_t**

Public Members

const char ***pathname**

lv_freetype_font_style_t **style**

lv_freetype_font_render_mode_t **render_mode**

uint32_t **ref_size**

Reference size for calculating outline glyph's real size.

FT_Face **face**

lv_mutex_t **face_lock**

bool **face_has_kerning**

lv_cache_t ***glyph_cache**

lv_cache_t ***draw_data_cache**

struct **_lv_freetype_context_t**

Public Members

FT_Library **library**

lv_ll_t **face_id_ll**

lv_event_cb_t **event_cb**

uint32_t **max_glyph_cnt**

lv_cache_t ***cache_node_cache**

struct **_lv_freetype_font_dsc_t**

Public Members

uint32_t **magic_num**

lv_font_t **font**

uint32_t **size**

lv_freetype_font_style_t **style**

lv_freetype_font_render_mode_t **render_mode**

lv_freetype_context_t ***context**

lv_freetype_cache_node_t ***cache_node**

lv_cache_entry_t ***cache_node_entry**

FTC_FaceID **face_id**

uint32_t **outline_stroke_width**

lv_font_kerning_t **kerning**

14.17.6 fsdrv

lv_fsdrv.h

Defines

LV_FS_MAX_PATH_LEN

Functions

void **lv_fs_fatfs_init**(void)

void **lv_fs_stdio_init**(void)

void **lv_fs_posix_init**(void)

void **lv_fs_win32_init**(void)

void **lv_fs_memfs_init**(void)

void **lv_littlefs_set_handler**(struct lfs*)

void **lv_fs_littlefs_init**(void)

void **lv_fs_arduino_esp_littlefs_init**(void)

void **lv_fs_arduino_sd_init**(void)

void **lv_fs_uefi_init**(void)

14.17.7 gif

lv_gif.h

Functions

lv_obj_t ***lv_gif_create**(lv_obj_t *parent)

Create a gif object

Parametri

parent -- pointer to an object, it will be the parent of the new gif.

Ritorna

pointer to the gif obj

void **lv_gif_set_src**(lv_obj_t *obj, const void *src)

Set the gif data to display on the object

Parametri

- **obj** -- pointer to a gif object
- **src** -- 1) pointer to an *lv_image_dsc_t* descriptor (which contains gif raw data) or 2) path to a gif file (e.g. "S:/dir/anim.gif")

void **lv_gif_restart**(lv_obj_t *obj)

Restart a gif animation.

Parametri

obj -- pointer to a gif obj

void **lv_gif_pause**(lv_obj_t *obj)

Pause a gif animation.

Parametri

obj -- pointer to a gif obj

void **lv_gif_resume**(lv_obj_t *obj)

Resume a gif animation.

Parametri

obj -- pointer to a gif obj

bool **lv_gif_is_loaded**(lv_obj_t *obj)

Checks if the GIF was loaded correctly.

Parametri

obj -- pointer to a gif obj

int32_t **lv_gif_get_loop_count**(lv_obj_t *obj)

Get the loop count for the GIF.

Parametri

obj -- pointer to a gif obj

void **lv_gif_set_loop_count**(lv_obj_t *obj, int32_t count)

Set the loop count for the GIF.

Parametri

- **obj** -- pointer to a gif obj
- **count** -- the loop count to set

Variables

const lv_obj_class_t **lv_gif_class**

lv_gif_private.h

struct **_lv_gif_t**

Public Members

lv_image_t **img**

gd_GIF ***gif**

lv_timer_t ***timer**

lv_image_dsc_t **imgdsc**

uint32_t **last_call**

14.17.8 libjpeg_turbo

lv_libjpeg_turbo.h

Functions

void **lv_libjpeg_turbo_init**(void)

Register the JPEG-Turbo decoder functions in LVGL

void **lv_libjpeg_turbo_deinit**(void)

14.17.9 libpng

lv_libpng.h

Functions

void **lv_libpng_init**(void)

Register the PNG decoder functions in LVGL

void **lv_libpng_deinit**(void)

14.17.10 lodepng

lv_lodepng.h

Functions

void **lv_lodepng_init**(void)

Register the PNG decoder functions in LVGL

void **lv_lodepng_deinit**(void)

14.17.11 qrcode

lv_qrcode.h

Functions

lv_obj_t ***lv_qrcode_create**(lv_obj_t *parent)

Create an empty QR code (an `lv_canvas`) object.

Parametri

parent -- point to an object where to create the QR code

Ritorna

pointer to the created QR code object

void **lv_qrcode_set_size**(lv_obj_t *obj, int32_t size)

Set QR code size.

Parametri

- **obj** -- pointer to a QR code object
- **size** -- width and height of the QR code

void **lv_qrcode_set_dark_color**(lv_obj_t *obj, *lv_color_t* color)

Set QR code dark color.

Parametri

- **obj** -- pointer to a QR code object
- **color** -- dark color of the QR code

void **lv_qrcode_set_light_color**(lv_obj_t *obj, *lv_color_t* color)

Set QR code light color.

Parametri

- **obj** -- pointer to a QR code object
- **color** -- light color of the QR code

lv_result_t **lv_qrcode_update**(lv_obj_t *obj, const void *data, uint32_t data_len)

Set the data of a QR code object

Parametri

- **obj** -- pointer to a QR code object
- **data** -- data to display
- **data_len** -- length of data in bytes

Ritorna

LV_RESULT_OK: if no error; LV_RESULT_INVALID: on error

Variables

const lv_obj_class_t **lv_qrcode_class**

lv_qrcode_private.h

struct **_lv_qrcode_t**

Public Members

lv_canvas_t **canvas**

lv_color_t **dark_color**

lv_color_t **light_color**

14.17.12 rle

lv_rle.h

Functions

uint32_t **lv_rle_decompress**(const uint8_t *input, uint32_t input_buff_len, uint8_t *output, uint32_t output_buff_len, uint8_t blk_size)

14.17.13 rlottie

lv_rlottie.h

Enums

enum **lv_rlottie_ctrl_t**

Values:

enumerator **LV_RLOTTIE_CTRL_FORWARD**

enumerator **LV_RLOTTIE_CTRL_BACKWARD**

enumerator **LV_RLOTTIE_CTRL_PAUSE**

enumerator **LV_RLOTTIE_CTRL_PLAY**

enumerator **LV_RLOTTIE_CTRL_LOOP**

Functions

lv_obj_t ***lv_rlottie_create_from_file**(lv_obj_t *parent, int32_t width, int32_t height, const char *path)

lv_obj_t ***lv_rlottie_create_from_raw**(lv_obj_t *parent, int32_t width, int32_t height, const char *rlottie_desc)

void **lv_rlottie_set_play_mode**(lv_obj_t *rlottie, const *lv_rlottie_ctrl_t* ctrl)

void **lv_rlottie_set_current_frame**(lv_obj_t *rlottie, const size_t goto_frame)

Variables

const lv_obj_class_t **lv_rlottie_class**

lv_rlottie_private.h

struct **_lv_rlottie_t**

Public Members

lv_image_t **img_ext**

struct Lottie_Animation_S ***animation**

lv_timer_t ***task**

lv_image_dsc_t **imgdsc**

size_t **total_frames**

size_t **current_frame**

size_t **framerate**

uint32_t ***allocated_buf**

size_t **allocated_buffer_size**

size_t **scanline_width**

lv_rlottie_ctrl_t **play_ctrl**

size_t **dest_frame**

14.17.14 svg

lv_svg.h

Defines

LV_SVG_NODE_CHILD(n, i)

LV_SVG_NODE(n)

Typedef

typedef int8_t **lv_svg_tag_t**

typedef uint8_t **lv_svg_attr_type_t**

typedef uint8_t **lv_svg_transform_type_t**

typedef uint32_t **lv_svg_aspect_ratio_t**

typedef uint32_t **lv_svg_color_t**

typedef uint8_t **lv_svg_fill_rule_t**

typedef uint8_t **lv_svg_line_cap_t**

typedef uint8_t **lv_svg_line_join_t**

typedef uint8_t **lv_svg_gradient_units_t**

typedef uint8_t **lv_svg_attr_value_type_t**

typedef uint8_t **lv_svg_attr_value_class_t**

Enums

enum **_lv_svg_tag_t**

Values:

enumerator **LV_SVG_TAG_INVALID**

enumerator **LV_SVG_TAG_CONTENT**

enumerator **LV_SVG_TAG_SVG**

enumerator **LV_SVG_TAG_USE**

enumerator **LV_SVG_TAG_G**

enumerator **LV_SVG_TAG_PATH**

enumerator **LV_SVG_TAG_RECT**

enumerator **LV_SVG_TAG_CIRCLE**

enumerator **LV_SVG_TAG_ELLIPSE**

enumerator **LV_SVG_TAG_LINE**

enumerator **LV_SVG_TAG_POLYLINE**

enumerator **LV_SVG_TAG_POLYGON**

enumerator **LV_SVG_TAG_SOLID_COLOR**

enumerator **LV_SVG_TAG_LINEAR_GRADIENT**

enumerator **LV_SVG_TAG_RADIAL_GRADIENT**

enumerator **LV_SVG_TAG_STOP**

enumerator **LV_SVG_TAG_DEFS**

enumerator **LV_SVG_TAG_IMAGE**

enumerator **LV_SVG_TAG_MPATH**

enumerator **LV_SVG_TAG_SET**

enumerator **LV_SVG_TAG_ANIMATE**

enumerator **LV_SVG_TAG_ANIMATE_COLOR**

enumerator **LV_SVG_TAG_ANIMATE_TRANSFORM**

enumerator **LV_SVG_TAG_ANIMATE_MOTION**

enumerator **LV_SVG_TAG_TEXT**

enumerator **LV_SVG_TAG_TSPAN**

enumerator **LV_SVG_TAG_TEXT_AREA**

enum **_lv_svg_attr_type_t**

Values:

enumerator **LV_SVG_ATTR_INVALID**

enumerator **LV_SVG_ATTR_ID**

enumerator **LV_SVG_ATTR_XML_ID**

enumerator **LV_SVG_ATTR_VERSION**

enumerator **LV_SVG_ATTR_BASE_PROFILE**

enumerator **LV_SVG_ATTR_VIEWBOX**

enumerator **LV_SVG_ATTR_PRESERVE_ASPECT_RATIO**

enumerator **LV_SVG_ATTR_VIEWPORT_FILL**

enumerator **LV_SVG_ATTR_VIEWPORT_FILL_OPACITY**

enumerator **LV_SVG_ATTR_DISPLAY**

enumerator **LV_SVG_ATTR_VISIBILITY**

enumerator **LV_SVG_ATTR_X**

enumerator **LV_SVG_ATTR_Y**

enumerator **LV_SVG_ATTR_WIDTH**

enumerator **LV_SVG_ATTR_HEIGHT**

enumerator **LV_SVG_ATTR_RX**

enumerator **LV_SVG_ATTR_RY**

enumerator **LV_SVG_ATTR_CX**

enumerator **LV_SVG_ATTR_CY**

enumerator **LV_SVG_ATTR_R**

enumerator **LV_SVG_ATTR_X1**

enumerator **LV_SVG_ATTR_Y1**

enumerator **LV_SVG_ATTR_X2**

enumerator **LV_SVG_ATTR_Y2**

enumerator **LV_SVG_ATTR_POINTS**

enumerator **LV_SVG_ATTR_D**

enumerator **LV_SVG_ATTR_PATH_LENGTH**

enumerator **LV_SVG_ATTR_XLINK_HREF**

enumerator **LV_SVG_ATTR_FILL**

enumerator **LV_SVG_ATTR_FILL_RULE**

enumerator **LV_SVG_ATTR_FILL_OPACITY**

enumerator **LV_SVG_ATTR_STROKE**

enumerator **LV_SVG_ATTR_STROKE_WIDTH**

enumerator **LV_SVG_ATTR_STROKE_LINECAP**

enumerator **LV_SVG_ATTR_STROKE_LINEJOIN**

enumerator **LV_SVG_ATTR_STROKE_MITER_LIMIT**

enumerator **LV_SVG_ATTR_STROKE_DASH_ARRAY**

enumerator **LV_SVG_ATTR_STROKE_DASH_OFFSET**

enumerator **LV_SVG_ATTR_STROKE_OPACITY**

enumerator **LV_SVG_ATTR_OPACITY**

enumerator **LV_SVG_ATTR_SOLID_COLOR**

enumerator **LV_SVG_ATTR_SOLID_OPACITY**

enumerator **LV_SVG_ATTR_GRADIENT_UNITS**

enumerator **LV_SVG_ATTR_GRADIENT_STOP_OFFSET**

enumerator **LV_SVG_ATTR_GRADIENT_STOP_COLOR**

enumerator **LV_SVG_ATTR_GRADIENT_STOP_OPACITY**

enumerator **LV_SVG_ATTR_FONT_FAMILY**

enumerator **LV_SVG_ATTR_FONT_STYLE**

enumerator **LV_SVG_ATTR_FONT_VARIANT**

enumerator **LV_SVG_ATTR_FONT_WEIGHT**

enumerator **LV_SVG_ATTR_FONT_SIZE**

enumerator **LV_SVG_ATTR_TRANSFORM**

enumerator **LV_SVG_ATTR_TEXT_ANCHOR**

enumerator **LV_SVG_ATTR_ATTRIBUTE_NAME**

enumerator **LV_SVG_ATTR_ATTRIBUTE_TYPE**

enumerator **LV_SVG_ATTR_BEGIN**

enumerator **LV_SVG_ATTR_END**

enumerator **LV_SVG_ATTR_DUR**

enumerator **LV_SVG_ATTR_MIN**

enumerator **LV_SVG_ATTR_MAX**

enumerator **LV_SVG_ATTR_RESTART**

enumerator **LV_SVG_ATTR_REPEAT_COUNT**

enumerator **LV_SVG_ATTR_REPEAT_DUR**

enumerator **LV_SVG_ATTR_CALC_MODE**

enumerator **LV_SVG_ATTR_VALUES**

enumerator **LV_SVG_ATTR_KEY_TIMES**

enumerator **LV_SVG_ATTR_KEY_SPLINES**

enumerator **LV_SVG_ATTR_KEY_POINTS**

enumerator **LV_SVG_ATTR_FROM**

enumerator **LV_SVG_ATTR_TO**

enumerator **LV_SVG_ATTR_BY**

enumerator **LV_SVG_ATTR_ADDITIVE**

enumerator **LV_SVG_ATTR_ACCUMULATE**

enumerator **LV_SVG_ATTR_PATH**

enumerator **LV_SVG_ATTR_ROTATE**

enumerator **LV_SVG_ATTR_TRANSFORM_TYPE**

enum **_lv_svg_transform_type_t**

Values:

enumerator **LV_SVG_TRANSFORM_TYPE_MATRIX**

enumerator **LV_SVG_TRANSFORM_TYPE_TRANSLATE**

enumerator **LV_SVG_TRANSFORM_TYPE_ROTATE**

enumerator **LV_SVG_TRANSFORM_TYPE_SCALE**

enumerator **LV_SVG_TRANSFORM_TYPE_SKEW_X**

enumerator **LV_SVG_TRANSFORM_TYPE_SKEW_Y**

enum **lv_svg_anim_action_t**

Values:

enumerator **LV_SVG_ANIM_REMOVE**

enumerator **LV_SVG_ANIM_FREEZE**

enum **_lv_svg_anim_restart_type_t**

Values:

enumerator **LV_SVG_ANIM_RESTART_ALWAYS**

enumerator **LV_SVG_ANIM_RESTART_WHEN_NOT_ACTIVE**

enumerator **LV_SVG_ANIM_RESTART_NEVER**

enum **_lv_svg_anim_calc_mode_t**

Values:

enumerator **LV_SVG_ANIM_CALC_MODE_LINEAR**

enumerator **LV_SVG_ANIM_CALC_MODE_PACED**

enumerator **LV_SVG_ANIM_CALC_MODE_SPLINE**

enumerator **LV_SVG_ANIM_CALC_MODE_DISCRETE**

enum **_lv_svg_anim_additive_type_t**

Values:

enumerator **LV_SVG_ANIM_ADDITIVE_REPLACE**

enumerator **LV_SVG_ANIM_ADDITIVE_SUM**

enum **_lv_svg_anim_accumulate_type_t**

Values:

enumerator **LV_SVG_ANIM_ACCUMULATE_NONE**

enumerator **LV_SVG_ANIM_ACCUMULATE_SUM**

enum **_lv_svg_aspect_ratio_t**

Values:

enumerator **LV_SVG_ASPECT_RATIO_NONE**

enumerator **LV_SVG_ASPECT_RATIO_XMIN_YMIN**

enumerator **LV_SVG_ASPECT_RATIO_XMID_YMIN**

enumerator **LV_SVG_ASPECT_RATIO_XMAX_YMIN**

enumerator **LV_SVG_ASPECT_RATIO_XMIN_YMID**

enumerator **LV_SVG_ASPECT_RATIO_XMID_YMID**

enumerator **LV_SVG_ASPECT_RATIO_XMAX_YMID**

enumerator **LV_SVG_ASPECT_RATIO_XMIN_YMAX**

enumerator **LV_SVG_ASPECT_RATIO_XMID_YMAX**

enumerator **LV_SVG_ASPECT_RATIO_XMAX_YMAX**

enum **_lv_svg_aspect_ratio_opt_t**

Values:

enumerator **LV_SVG_ASPECT_RATIO_OPT_MEET**

enumerator **LV_SVG_ASPECT_RATIO_OPT_SLICE**

enum **_lv_svg_fill_rule_t**

Values:

enumerator **LV_SVG_FILL_NONZERO**

enumerator **LV_SVG_FILL_EVENODD**

enum **_lv_svg_line_cap_t**

Values:

enumerator **LV_SVG_LINE_CAP_BUTT**

enumerator **LV_SVG_LINE_CAP_SQUARE**

enumerator **LV_SVG_LINE_CAP_ROUND**

enum **_lv_svg_line_join_t**

Values:

enumerator **LV_SVG_LINE_JOIN_MITER**

enumerator **LV_SVG_LINE_JOIN_BEVEL**

enumerator **LV_SVG_LINE_JOIN_ROUND**

enum **_lv_svg_gradient_units_t**

Values:

enumerator **LV_SVG_GRADIENT_UNITS_OBJECT**

enumerator **LV_SVG_GRADIENT_UNITS_USER_SPACE**

enum **_lv_svg_path_cmd_t**

Values:

enumerator **LV_SVG_PATH_CMD_MOVE_TO**

enumerator **LV_SVG_PATH_CMD_LINE_TO**

enumerator **LV_SVG_PATH_CMD_CURVE_TO**

enumerator **LV_SVG_PATH_CMD_QUAD_TO**

enumerator **LV_SVG_PATH_CMD_ARC_TO**

enumerator **LV_SVG_PATH_CMD_CLOSE**

enum **_lv_svg_attr_value_type_t**

Values:

enumerator **LV_SVG_ATTR_VALUE_DATA**

enumerator **LV_SVG_ATTR_VALUE_PTR**

enum **_lv_svg_attr_value_class_t**

Values:

enumerator **LV_SVG_ATTR_VALUE_NONE**

enumerator **LV_SVG_ATTR_VALUE_INITIAL**

enumerator **LV_SVG_ATTR_VALUE_INHERIT**

Functions

lv_svg_node_t ***lv_svg_load_data**(const char *svg_data, uint32_t data_len)

Loading SVG data and creating the DOM tree.

Parametri

- **svg_data** -- pointer to the SVG data
- **data_len** -- the SVG data length

lv_svg_node_t ***lv_svg_node_create**(*lv_svg_node_t* *parent)

Create an SVG DOM node.

Parametri

parent -- pointer to the parent node

Ritorna

true: an new SVG DOM node, false: NULL

void **lv_svg_node_delete**(*lv_svg_node_t* *node)

Delete an SVG DOM subtree.

Parametri

node -- pointer to an SVG DOM subtree

struct **lv_svg_point_t**

Public Members

float **x**

float **y**

struct **lv_svg_matrix_t**

Public Members

float **m**[3][3]

union **lv_svg_attr_value_t**

Public Members

int32_t **ival**

uint32_t **uval**

float **fval**

char ***sval**

void ***val**

struct **lv_svg_attr_values_list_t**

Public Members

uint32_t **length**

uint8_t **data**[1]

struct **lv_svg_attr_path_value_t**

Public Members

uint32_t **cmd**

uint8_t **data**[1]

struct **lv_svg_attr_t**

Public Members

lv_svg_attr_type_t **id**

lv_svg_attr_value_type_t **val_type**

lv_svg_attr_value_class_t **class_type**

lv_svg_attr_value_t **value**

struct **lv_svg_node_t**

Public Members

lv_tree_node_t **base**

char ***xml_id**

lv_svg_tag_t **type**

lv_array_t **attrs**

struct *_lv_svg_render_obj* ***render_obj**

lv_svg_decoder.h

Functions

void **lv_svg_decoder_init**(void)

Register the SVG decoder functions in LVGL

void **lv_svg_decoder_deinit**(void)

lv_svg_parser.h

Enums

enum **_lv_svg_parser_state_t**

Values:

enumerator **LV_SVG_PARSER_PROCESS**

enumerator **LV_SVG_PARSER_IGNORE**

Functions

void **_lv_svg_parser_init**(*_lv_svg_parser_t* *parser)

Initialize the SVG parser.

Parametri

parser -- pointer to a parser object

void **_lv_svg_parser_deinit**(*_lv_svg_parser_t* *parser)

Deinitialize the SVG parser.

Parametri

parser -- pointer to a parser object

bool **_lv_svg_parser_token**(*_lv_svg_parser_t* *parser, const *_lv_svg_token_t* *token)

Parse an SVG document.

Parametri

- **parser** -- pointer to a parser object
- **token** -- pointer to a token object

Ritorna

true: the parsing is finished, false: the parsing is not finished yet.

bool **_lv_svg_parser_is_finish**(*_lv_svg_parser_t* *parser)

Check if the parsing is finished.

Parametri

parser -- pointer to a parser object

Ritorna

true: the parsing is finished, false: the parsing is not finished yet.

void **_lv_svg_dump_tree**(*lv_svg_node_t* *root, int depth)

Dump the SVG tree.

Parametri

- **root** -- pointer to the root of the SVG tree
- **depth** -- the depth of the current node in the tree

struct **_lv_svg_parser_t**

Public Members

uint16_t **state**

char ***ignore_name**

uint32_t **ignore_len**

int32_t **dpi**

lv_svg_node_t ***doc_root**

lv_svg_node_t ***cur_node**

lv_svg_render.h

Defines

LV_SVG_RENDER_OBJ(n)

Typedef

typedef struct *_lv_svg_render_obj* **lv_svg_render_obj_t**

typedef struct *_lv_svg_render_class* **lv_svg_render_class**

typedef struct *_lv_svg_render_hal* **lv_svg_render_hal_t**

Functions

void **lv_svg_render_init**(const *lv_svg_render_hal_t* *hal)

Initialize the SVG render.

Parametri

hal -- pointer to a structure with rendering functions

lv_svg_render_obj_t ***lv_svg_render_create**(const *lv_svg_node_t* *svg_doc)

Create a new SVG render from an SVG document.

Parametri

svg_doc -- pointer to the SVG document

Ritorna

pointer to the new SVG render object

void **lv_svg_render_delete**(*lv_svg_render_obj_t* *render)

Delete an SVG render object.

Parametri

render -- pointer to the SVG render object to delete

uint32_t **lv_svg_render_get_size**(const *lv_svg_render_obj_t* *render)

Get size of render objects.

Parametri

render -- pointer to the SVG render object

Ritorna

the bytes of SVG render objects

lv_result_t **lv_svg_render_get_viewport_size**(const *lv_svg_render_obj_t* *render, float *width, float *height)

Get viewport's width and height of the render object.

Parametri

- **render** -- pointer to the SVG render object
- **width** -- pointer to save the width of the viewport of the SVG render object
- **height** -- pointer to save the height of the viewport of the SVG render object

Ritorna

lv_result_t, LV_RESULT_OK if success, LV_RESULT_INVALID if fail

void **lv_draw_svg_render**(lv_vector_dsc_t *dsc, const *lv_svg_render_obj_t* *render)

Render an SVG object to a vector graphics.

Parametri

- **dsc** -- pointer to the vector graphics descriptor
- **render** -- pointer to the SVG render object to render

void **lv_draw_svg**(lv_layer_t *layer, const *lv_svg_node_t* *svg_doc)

Draw an SVG document to a layer.

Parametri

- **layer** -- pointer to the target layer
- **svg_doc** -- pointer to the SVG document to draw

struct **_lv_svg_render_obj**

Public Members

struct *_lv_svg_render_obj* ***next**

lv_svg_tag_t **tag**

uint32_t **flags**

char ***id**

lv_vector_draw_dsc_t **dsc**

lv_matrix_t **matrix**

struct *_lv_svg_render_obj* ***head**

char ***fill_ref**

char ***stroke_ref**

struct *_lv_svg_render_class* ***clz**

struct **_lv_svg_render_class**

Public Members

void (*`set_paint_ref`)(struct *_lv_svg_render_obj* *obj, lv_vector_draw_dsc_t *dsc, const struct *_lv_svg_render_obj* *target_obj, bool fill)

void (*`init`)(struct *_lv_svg_render_obj* *obj, const *lv_svg_node_t* *node)

void (*`render`)(const struct *_lv_svg_render_obj* *obj, lv_vector_dsc_t *dsc, const lv_matrix_t *matrix)

void (*`set_attr`)(struct *_lv_svg_render_obj* *obj, lv_vector_draw_dsc_t *dsc, const *lv_svg_attr_t* *attr)

void (*`get_bounds`)(const struct *_lv_svg_render_obj* *obj, *lv_area_t* *area)

void (*`get_size`)(const struct *_lv_svg_render_obj* *obj, uint32_t *size)

void (*`destroy`)(struct *_lv_svg_render_obj* *obj)

struct **`_lv_svg_render_hal`**

Public Members

void (*`load_image`)(const char *image_url, lv_draw_image_dsc_t *img_dsc)

const char *(*`get_font_path`)(const char *font_family)

lv_svg_token.h

Defines

`SVG_TOKEN_LEN``(t)`

Typedef

typedef bool (*`svg_token_process`)(*_lv_svg_token_t* *token, void *user_data)

Enums

enum **`_lv_svg_token_type_t`**

Values:

enumerator **`LV_SVG_TOKEN_BEGIN`**

enumerator **`LV_SVG_TOKEN_END`**

enumerator **`LV_SVG_TOKEN_CONTENT`**

Functions

bool **_lv_svg_tokenizer**(const char *svg_data, uint32_t len, *svg_token_process* cb, void *user_data)

Parse SVG data and call a callback for each token.

Parametri

- **svg_data** -- pointer to SVG data
- **len** -- length of the SVG data
- **cb** -- callback function to be called for each token
- **user_data** -- custom data to be passed to the callback function

Ritorna

true: SVG data successfully parsed, false: error occurred

struct **_lv_svg_token_attr_t**

Public Members

const char ***name_start**

const char ***name_end**

const char ***value_start**

const char ***value_end**

struct **_lv_svg_token_t**

Public Members

const char ***start**

const char ***end**

_lv_svg_token_type_t **type**

bool **flat**

_lv_svg_token_attr_t ***cur_attr**

lv_array_t **attrs**

14.17.15 tiny_ttf

lv_tiny_ttf.h

Functions

lv_font_t ***lv_tiny_ttf_create_data**(const void *data, size_t data_size, int32_t font_size)

Create a font from the specified data pointer with the specified line height.

Parametri

- **data** -- the data pointer
- **data_size** -- the data size
- **font_size** -- the font size in pixel

Ritorna
a font object

lv_font_t *`lv_tiny_ttf_create_data_ex`(const void *data, size_t data_size, int32_t font_size, *lv_font_kerning_t* kerning, size_t cache_size)

Create a font from the specified data pointer with the specified line height and the specified cache size.

Parametri
- **data** -- the data pointer
- **data_size** -- the data size
- **font_size** -- the font size in pixel
- **kerning** -- kerning value in pixel
- **cache_size** -- the cache size in count

Ritorna

void `lv_tiny_ttf_set_size`(lv_font_t *font, int32_t font_size)

Set the size of the font to a new font_size

> **Nota**
> the font bitmap cache and glyph cache will be flushed.

Parametri
- **font** -- the font object
- **font_size** -- the font size in pixel

void `lv_tiny_ttf_destroy`(lv_font_t *font)

Destroy a font previously created with lv_tiny_ttf_create_xxxx()

Parametri
font -- the font object

Variables

const lv_font_class_t `lv_tiny_ttf_font_class`

struct `lv_tiny_ttf_font_src_t`

Public Members

const char *`path`

Path to the font file

const void *`data`

Pointer to the font data

size_t `data_size`

Size of the font data

size_t `cache_size`

Size of the font cache

14.17.16 tjpgd

lv_tjpgd.h

14.18 misc

14.18.1 lv_anim.h

Defines

LV_ANIM_REPEAT_INFINITE

LV_ANIM_PLAYTIME_INFINITE

LV_ANIM_PAUSE_FOREVER

_PARA(a, x1, y1, x2, y2)

LV_ANIM_SET_EASE_IN_SINE(a)

LV_ANIM_SET_EASE_OUT_SINE(a)

LV_ANIM_SET_EASE_IN_OUT_SINE(a)

LV_ANIM_SET_EASE_IN_QUAD(a)

LV_ANIM_SET_EASE_OUT_QUAD(a)

LV_ANIM_SET_EASE_IN_OUT_QUAD(a)

LV_ANIM_SET_EASE_IN_CUBIC(a)

LV_ANIM_SET_EASE_OUT_CUBIC(a)

LV_ANIM_SET_EASE_IN_OUT_CUBIC(a)

LV_ANIM_SET_EASE_IN_QUART(a)

LV_ANIM_SET_EASE_OUT_QUART(a)

LV_ANIM_SET_EASE_IN_OUT_QUART(a)

LV_ANIM_SET_EASE_IN_QUINT(a)

LV_ANIM_SET_EASE_OUT_QUINT(a)

LV_ANIM_SET_EASE_IN_OUT_QUINT(a)

LV_ANIM_SET_EASE_IN_EXPO(a)

LV_ANIM_SET_EASE_OUT_EXPO(a)

LV_ANIM_SET_EASE_IN_OUT_EXPO(a)

LV_ANIM_SET_EASE_IN_CIRC(a)

LV_ANIM_SET_EASE_OUT_CIRC(a)

LV_ANIM_SET_EASE_IN_OUT_CIRC(a)

LV_ANIM_SET_EASE_IN_BACK(a)

LV_ANIM_SET_EASE_OUT_BACK(a)

LV_ANIM_SET_EASE_IN_OUT_BACK(a)

LV_ANIM_OFF

Can be used to indicate if animations are enabled or disabled in a case

LV_ANIM_ON

Typedef

typedef bool **lv_anim_enable_t**

typedef int32_t (*__lv_anim_path_cb_t__)(const lv_anim_t*)

Get the current value during an animation

typedef void (*__lv_anim_exec_xcb_t__)(void*, int32_t)

Generic prototype of "animator" functions. First parameter is the variable to animate. Second parameter is the value to set. Compatible with `lv_xxx_set_yyy(obj, value)` functions The `x` in `_xcb_t` means it's not a fully generic prototype because it doesn't receive `lv_anim_t *` as its first argument

typedef void (*__lv_anim_custom_exec_cb_t__)(lv_anim_t*, int32_t)

Same as *`lv_anim_exec_xcb_t`* but receives `lv_anim_t *` as the first parameter. It's more consistent but less convenient. Might be used by binding generator functions.

typedef void (*__lv_anim_completed_cb_t__)(lv_anim_t*)

Callback to call when the animation is ready

typedef void (*__lv_anim_start_cb_t__)(lv_anim_t*)

Callback to call when the animation really stars (considering `delay`)

typedef int32_t (*__lv_anim_get_value_cb_t__)(lv_anim_t*)

Callback used when the animation values are relative to get the current value

typedef void (*__lv_anim_deleted_cb_t__)(lv_anim_t*)

Callback used when the animation is deleted

Functions

void **lv_anim_init**(lv_anim_t *a)

Initialize an animation variable. E.g.: lv_anim_t a; lv_anim_init(&a); lv_anim_set_...(&a); lv_anim_start(&a);

Parametri

a -- pointer to an `lv_anim_t` variable to initialize

void **lv_anim_set_var**(lv_anim_t *a, void *var)

Set a variable to animate

Parametri

- **a** -- pointer to an initialized `lv_anim_t` variable
- **var** -- pointer to a variable to animate

void **lv_anim_set_exec_cb**(lv_anim_t *a, *lv_anim_exec_xcb_t* exec_cb)

Set a function to animate `var`

Parametri

- **a** -- pointer to an initialized `lv_anim_t` variable

- **exec_cb** -- a function to execute during animation LVGL's built-in functions can be used. E.g. lv_obj_set_x

void **lv_anim_set_duration**(lv_anim_t *a, uint32_t duration)

Set the duration of an animation

Parametri

- **a** -- pointer to an initialized `lv_anim_t` variable
- **duration** -- duration of the animation in milliseconds

void **lv_anim_set_delay**(lv_anim_t *a, uint32_t delay)

Set a delay before starting the animation

Parametri

- **a** -- pointer to an initialized `lv_anim_t` variable
- **delay** -- delay before the animation in milliseconds

void **lv_anim_resume**(lv_anim_t *a)

Resumes a paused animation

Parametri

a -- pointer to an initialized `lv_anim_t` variable

void **lv_anim_pause**(lv_anim_t *a)

Pauses the animation

Parametri

a -- pointer to an initialized `lv_anim_t` variable

void **lv_anim_pause_for**(lv_anim_t *a, uint32_t ms)

Pauses the animation for ms milliseconds

Parametri

- **a** -- pointer to an initialized `lv_anim_t` variable
- **ms** -- the pause time in milliseconds

bool **lv_anim_is_paused**(lv_anim_t *a)

Check if the animation is paused

Parametri

a -- pointer to an initialized `lv_anim_t` variable

Ritorna

true if the animation is paused else false

void **lv_anim_set_values**(lv_anim_t *a, int32_t start, int32_t end)

Set the start and end values of an animation

Parametri

- **a** -- pointer to an initialized `lv_anim_t` variable
- **start** -- the start value
- **end** -- the end value

void **lv_anim_set_custom_exec_cb**(lv_anim_t *a, *lv_anim_custom_exec_cb_t* exec_cb)

Similar to `lv_anim_set_exec_cb` but *`lv_anim_custom_exec_cb_t`* receives `lv_anim_t *` as its first parameter instead of `void *`. This function might be used when LVGL is bound to other languages because it's more consistent to have `lv_anim_t *` as first parameter.

Parametri

- **a** -- pointer to an initialized `lv_anim_t` variable
- **exec_cb** -- a function to execute.

void **lv_anim_set_path_cb**(lv_anim_t *a, *lv_anim_path_cb_t* path_cb)

Set the path (curve) of the animation.

Parametri

- **a** -- pointer to an initialized `lv_anim_t` variable
- **path_cb** -- a function to set the current value of the animation.

void **lv_anim_set_start_cb**(lv_anim_t *a, *lv_anim_start_cb_t* start_cb)

Set a function call when the animation really starts (considering `delay`)

Parametri

- **a** -- pointer to an initialized `lv_anim_t` variable
- **start_cb** -- a function call when the animation starts

void **lv_anim_set_get_value_cb**(lv_anim_t *a, *lv_anim_get_value_cb_t* get_value_cb)

Set a function to use the current value of the variable and make start and end value relative to the returned current value.

Parametri

- **a** -- pointer to an initialized `lv_anim_t` variable
- **get_value_cb** -- a function call when the animation starts

void **lv_anim_set_completed_cb**(lv_anim_t *a, *lv_anim_completed_cb_t* completed_cb)

Set a function call when the animation is completed

Parametri

- **a** -- pointer to an initialized `lv_anim_t` variable
- **completed_cb** -- a function call when the animation is fully completed

void **lv_anim_set_deleted_cb**(lv_anim_t *a, *lv_anim_deleted_cb_t* deleted_cb)

Set a function call when the animation is deleted.

Parametri

- **a** -- pointer to an initialized `lv_anim_t` variable
- **deleted_cb** -- a function call when the animation is deleted

void **lv_anim_set_reverse_duration**(lv_anim_t *a, uint32_t duration)

Make the animation to play back to when the forward direction is ready

Parametri

- **a** -- pointer to an initialized `lv_anim_t` variable
- **duration** -- duration of playback animation in milliseconds. 0: disable playback

void **lv_anim_set_reverse_time**(lv_anim_t *a, uint32_t duration)

Legacy `lv_anim_set_reverse_time` API will be removed soon, use `lv_anim_set_reverse_duration` instead.

void **lv_anim_set_reverse_delay**(lv_anim_t *a, uint32_t delay)

Make the animation to play back to when the forward direction is ready

Parametri

- **a** -- pointer to an initialized `lv_anim_t` variable
- **delay** -- delay in milliseconds before starting the playback animation.

void **lv_anim_set_repeat_count**(lv_anim_t *a, uint32_t cnt)

Make the animation repeat itself.

Parametri

- **a** -- pointer to an initialized `lv_anim_t` variable
- **cnt** -- repeat count or `LV_ANIM_REPEAT_INFINITE` for infinite repetition. 0: to disable repetition.

void **lv_anim_set_repeat_delay**(lv_anim_t *a, uint32_t delay)

Set a delay before repeating the animation.

Parametri

- **a** -- pointer to an initialized `lv_anim_t` variable
- **delay** -- delay in milliseconds before repeating the animation.

void **lv_anim_set_early_apply**(lv_anim_t *a, bool en)

Set a whether the animation's should be applied immediately or only when the delay expired.

Parametri

- **a** -- pointer to an initialized `lv_anim_t` variable
- **en** -- true: apply the start value immediately in `lv_anim_start`; false: apply the start value only when `delay` ms is elapsed and the animations really starts

void **lv_anim_set_user_data**(lv_anim_t *a, void *user_data)

Set the custom user data field of the animation.

Parametri

- **a** -- pointer to an initialized `lv_anim_t` variable
- **user_data** -- pointer to the new user_data.

void **lv_anim_set_bezier3_param**(lv_anim_t *a, int16_t x1, int16_t y1, int16_t x2, int16_t y2)

Set parameter for cubic bezier path

Parametri

- **a** -- pointer to an initialized `lv_anim_t` variable
- **x1** -- first control point X
- **y1** -- first control point Y
- **x2** -- second control point X
- **y2** -- second control point Y

lv_anim_t ***lv_anim_start**(const lv_anim_t *a)

Create an animation

Parametri

a -- an initialized 'anim_t' variable. Not required after call.

Ritorna

pointer to the created animation (different from the `a` parameter)

uint32_t **lv_anim_get_delay**(const lv_anim_t *a)

Get a delay before starting the animation

Parametri

a -- pointer to an initialized `lv_anim_t` variable

Ritorna

delay before the animation in milliseconds

uint32_t **lv_anim_get_playtime**(const lv_anim_t *a)

Get the time used to play the animation.

Parametri

a -- pointer to an animation.

Ritorna

the play time in milliseconds.

uint32_t **lv_anim_get_time**(const lv_anim_t *a)

Get the duration of an animation

Parametri

a -- pointer to an initialized `lv_anim_t` variable

Ritorna

the duration of the animation in milliseconds

uint32_t **lv_anim_get_repeat_count**(const lv_anim_t *a)

Get the repeat count of the animation.

Parametri

a -- pointer to an initialized `lv_anim_t` variable

Ritorna

the repeat count or `LV_ANIM_REPEAT_INFINITE` for infinite repetition. 0: disabled repetition.

void ***lv_anim_get_user_data**(const lv_anim_t *a)

Get the user_data field of the animation

Parametri

a -- pointer to an initialized `lv_anim_t` variable

Ritorna

the pointer to the custom user_data of the animation

bool **lv_anim_delete**(void *var, *lv_anim_exec_xcb_t* exec_cb)

Delete animation(s) of a variable with a given animator function

Parametri

- **var** -- pointer to variable

- **exec_cb** -- a function pointer which is animating 'var', or NULL to ignore it and delete all the animations of 'var

Ritorna

true: at least 1 animation is deleted, false: no animation is deleted

void **lv_anim_delete_all**(void)

Delete all the animations

lv_anim_t ***lv_anim_get**(void *var, *lv_anim_exec_xcb_t* exec_cb)

Get the animation of a variable and its `exec_cb`.

Parametri

- **var** -- pointer to variable
- **exec_cb** -- a function pointer which is animating 'var', or NULL to return first matching 'var'

Ritorna

pointer to the animation.

lv_timer_t ***lv_anim_get_timer**(void)

Get global animation refresher timer.

Ritorna

pointer to the animation refresher timer.

bool **lv_anim_custom_delete**(lv_anim_t *a, *lv_anim_custom_exec_cb_t* exec_cb)

Delete an animation by getting the animated variable from `a`. Only animations with `exec_cb` will be deleted. This function exists because it's logical that all anim. functions receives an `lv_anim_t` as their first parameter. It's not practical in C but might make the API more consequent and makes easier to generate bindings.

Parametri

- **a** -- pointer to an animation.
- **exec_cb** -- a function pointer which is animating 'var', or NULL to ignore it and delete all the animations of 'var

Ritorna

true: at least 1 animation is deleted, false: no animation is deleted

lv_anim_t ***lv_anim_custom_get**(lv_anim_t *a, *lv_anim_custom_exec_cb_t* exec_cb)

Get the animation of a variable and its `exec_cb`. This function exists because it's logical that all anim. functions receives an `lv_anim_t` as their first parameter. It's not practical in C but might make the API more consequent and makes easier to generate bindings.

Parametri

- **a** -- pointer to an animation.
- **exec_cb** -- a function pointer which is animating 'var', or NULL to return first matching 'var'

Ritorna

pointer to the animation.

uint16_t **lv_anim_count_running**(void)

Get the number of currently running animations

Ritorna

the number of running animations

uint32_t **lv_anim_speed**(uint32_t speed)

Store the speed as a special value which can be used as time in animations. It will be converted to time internally based on the start and end values. The return value can be used as a constant with multiple animations and let LVGL convert the speed to time based on the actual values. LIMITATION: the max time stored this way can be 10,000 ms.

> **Nota**
>
> internally speed is stored as 10 unit/sec

Parametri

speed -- the speed of the animation in with unit / sec resolution in 0..10k range

Ritorna

a special value which can be used as an animation time

uint32_t **lv_anim_speed_clamped**(uint32_t speed, uint32_t min_time, uint32_t max_time)

Store the speed as a special value which can be used as time in animations. It will be converted to time internally based on the start and end values. The return value can be used as a constant with multiple animations and let LVGL convert the speed to time based on the actual values.

> **Nota**
>
> internally speed is stored as 10 unit/sec

> **Nota**
>
> internally min/max_time are stored with 10 ms unit

Parametri

- **speed** -- the speed of the animation in as unit / sec resolution in 0..10k range
- **min_time** -- the minimum time in 0..10k range
- **max_time** -- the maximum time in 0..10k range

Ritorna

a special value in where all three values are stored and can be used as an animation time

uint32_t **lv_anim_resolve_speed**(uint32_t speed, int32_t start, int32_t end)

Resolve the speed (created with `lv_anim_speed` or `lv_anim_speed_clamped`) to time based on start and end values.

Parametri

- **speed** -- return values of `lv_anim_speed` or `lv_anim_speed_clamped`
- **start** -- the start value of the animation
- **end** -- the end value of the animation

Ritorna

the time required to get from `start` to `end` with the given `speed` setting

uint32_t **lv_anim_speed_to_time**(uint32_t speed, int32_t start, int32_t end)

Calculate the time of an animation based on its speed, start and end values. It simpler than `lv_anim_speed` or `lv_anim_speed_clamped` as it converts speed, start, and end to a time immediately. As it's simpler there is no limit on the maximum time.

Parametri

- **speed** -- the speed of the animation
- **start** -- the start value
- **end** -- the end value

Ritorna

the time of the animation in milliseconds

void **lv_anim_refr_now**(void)

Manually refresh the state of the animations. Useful to make the animations running in a blocking process where `lv_timer_handler` can't run for a while. Shouldn't be used directly because it is called in *`lv_refr_now()`*.

int32_t **lv_anim_path_linear**(const lv_anim_t *a)

Calculate the current value of an animation applying linear characteristic

Parametri

a -- pointer to an animation

Ritorna

the current value to set

int32_t **lv_anim_path_ease_in**(const lv_anim_t *a)

Calculate the current value of an animation slowing down the start phase

Parametri

a -- pointer to an animation

Ritorna

the current value to set

int32_t **lv_anim_path_ease_out**(const lv_anim_t *a)

Calculate the current value of an animation slowing down the end phase

Parametri

a -- pointer to an animation

Ritorna

the current value to set

int32_t **lv_anim_path_ease_in_out**(const lv_anim_t *a)

Calculate the current value of an animation applying an "S" characteristic (cosine)

Parametri

a -- pointer to an animation

Ritorna

the current value to set

int32_t **lv_anim_path_overshoot**(const lv_anim_t *a)

Calculate the current value of an animation with overshoot at the end

Parametri

a -- pointer to an animation

Ritorna

the current value to set

int32_t **lv_anim_path_bounce**(const lv_anim_t *a)

Calculate the current value of an animation with 3 bounces

Parametri

a -- pointer to an animation

Ritorna

the current value to set

int32_t **lv_anim_path_step**(const lv_anim_t *a)

Calculate the current value of an animation applying step characteristic. (Set end value on the end of the animation)

Parametri

a -- pointer to an animation

Ritorna

the current value to set

int32_t **lv_anim_path_custom_bezier3**(const lv_anim_t *a)

A custom cubic bezier animation path, need to specify cubic-parameters in a->parameter.bezier3

Parametri

a -- pointer to an animation

Ritorna

the current value to set

struct **lv_anim_bezier3_para_t**

#include <lv_anim.h> Parameter used when path is custom_bezier

Public Members

int16_t **x1**

int16_t **y1**

int16_t **x2**

int16_t **y2**

struct **_lv_anim_t**

#include <lv_anim.h> Describes an animation

Public Members

void ***var**

Variable (Widget or other user-provided object) to animate

lv_anim_exec_xcb_t **exec_cb**

Function to execute to animate

lv_anim_custom_exec_cb_t **custom_exec_cb**

Function to execute to animate, same purpose as exec_cb but different parameters

lv_anim_start_cb_t **start_cb**

Call it when animation is starts (considering `delay`)

lv_anim_completed_cb_t **completed_cb**

Call it when animation is fully completed

lv_anim_deleted_cb_t **deleted_cb**

Call it when animation is deleted

lv_anim_get_value_cb_t **get_value_cb**

Get current value in relative mode

void ***user_data**

Custom user data

lv_anim_path_cb_t **path_cb**

Provides path (curve) of animation

int32_t **start_value**

Start value

int32_t **current_value**

Current value

int32_t **end_value**

End value

int32_t **duration**

Animation duration in ms

int32_t **act_time**

Ms elapsed since animation started. Set to negative to make delay.

uint32_t **reverse_delay**

Wait (in ms) after forward play ends and before reverse play begins.

uint32_t **reverse_duration**

Reverse animation duration in ms

uint32_t **repeat_delay**

Wait before repeating

uint32_t **repeat_cnt**

Repeat count for animation

union *_lv_anim_t*::*_lv_anim_path_para_t* **parameter**

uint32_t **last_timer_run**

uint32_t **pause_time**

The time when the animation was paused

uint32_t **pause_duration**

The amount of the time the animation must stay paused for

uint8_t **is_paused**

Indicates that the animation is paused

uint8_t **reverse_play_in_progress**

Reverse play is in progress

uint8_t **run_round**

When not equal to global.anim_state.anim_run_round (which toggles each time animation timer executes), indicates this animation needs to be updated.

uint8_t **start_cb_called**

Indicates that `start_cb` was already called

uint8_t **early_apply**

1: Apply start value immediately even is there is a `delay`

union **_lv_anim_path_para_t**

Public Members

lv_anim_bezier3_para_t **bezier3**

Parameter used when path is custom_bezier

14.18.2 lv_anim_private.h

Functions

void **lv_anim_core_init**(void)

Init the animation module

void **lv_anim_core_deinit**(void)

Deinit the animation module

void **lv_anim_enable_vsync_mode**(bool enable)

struct **lv_anim_state_t**

Public Members

bool **anim_list_changed**

bool **anim_run_round**

bool **anim_vsync_registered**

lv_timer_t ***timer**

lv_ll_t **anim_ll**

14.18.3 lv_anim_timeline.h

Defines

LV_ANIM_TIMELINE_PROGRESS_MAX

Typedef

typedef struct _lv_anim_timeline_t **lv_anim_timeline_t**

Functions

lv_anim_timeline_t ***lv_anim_timeline_create**(void)

Create an animation timeline.

Ritorna

pointer to the animation timeline.

void **lv_anim_timeline_delete**(*lv_anim_timeline_t* *at)

Delete animation timeline.

Parametri

at -- pointer to the animation timeline.

void **lv_anim_timeline_add**(*lv_anim_timeline_t* *at, uint32_t start_time, const lv_anim_t *a)

Add animation to the animation timeline.

Parametri

- **at** -- pointer to the animation timeline.
- **start_time** -- the time the animation started on the timeline, note that start_time will override the value of delay.
- **a** -- pointer to an animation.

uint32_t **lv_anim_timeline_start**(*lv_anim_timeline_t* *at)

Start the animation timeline.

Parametri

at -- pointer to the animation timeline.

Ritorna

total time spent in animation timeline.

void **lv_anim_timeline_pause**(*lv_anim_timeline_t* *at)

Pause the animation timeline.

Parametri

at -- pointer to the animation timeline.

void **lv_anim_timeline_set_reverse**(*lv_anim_timeline_t* *at, bool reverse)

Set the playback direction of the animation timeline.

Parametri

- **at** -- pointer to the animation timeline.
- **reverse** -- whether to play in reverse.

void **lv_anim_timeline_set_repeat_count**(*lv_anim_timeline_t* *at, uint32_t cnt)

Make the animation timeline repeat itself.

Parametri

- **at** -- pointer to the animation timeline.
- **cnt** -- repeat count or `LV_ANIM_REPEAT_INFINITE` for infinite repetition. 0: to disable repetition.

void **lv_anim_timeline_set_repeat_delay**(*lv_anim_timeline_t* *at, uint32_t delay)

Set a delay before repeating the animation timeline.

Parametri

- **at** -- pointer to the animation timeline.
- **delay** -- delay in milliseconds before repeating the animation timeline.

void **lv_anim_timeline_set_progress**(*lv_anim_timeline_t* *at, uint16_t progress)

Set the progress of the animation timeline.

Parametri

- **at** -- pointer to the animation timeline.
- **progress** -- set value 0~65535 to map 0~100% animation progress.

uint32_t **lv_anim_timeline_get_playtime**(*lv_anim_timeline_t* *at)

Get the time used to play the animation timeline.

Parametri

at -- pointer to the animation timeline.

Ritorna

total time spent in animation timeline.

bool **lv_anim_timeline_get_reverse**(*lv_anim_timeline_t* *at)

Get whether the animation timeline is played in reverse.

Parametri

at -- pointer to the animation timeline.

Ritorna

return true if it is reverse playback.

uint16_t **lv_anim_timeline_get_progress**(*lv_anim_timeline_t* *at)

Get the progress of the animation timeline.

Parametri

at -- pointer to the animation timeline.

Ritorna

return value 0~65535 to map 0~100% animation progress.

uint32_t **lv_anim_timeline_get_repeat_count**(*lv_anim_timeline_t* *at)

Get repeat count of the animation timeline.

Parametri
at -- pointer to the animation timeline.

uint32_t **lv_anim_timeline_get_repeat_delay**(*lv_anim_timeline_t* *at)

Get repeat delay of the animation timeline.

Parametri
at -- pointer to the animation timeline.

14.18.4 lv_area.h

Defines

LV_COORD_TYPE_SHIFT

LV_COORD_TYPE_MASK

LV_COORD_TYPE(x)

LV_COORD_PLAIN(x)

LV_COORD_TYPE_PX

LV_COORD_TYPE_SPEC

LV_COORD_TYPE_PX_NEG

LV_COORD_IS_PX(x)

LV_COORD_IS_SPEC(x)

LV_COORD_SET_SPEC(x)

LV_COORD_MAX

Max coordinate value

LV_COORD_MIN

LV_SIZE_CONTENT

LV_PCT_STORED_MAX

LV_PCT_POS_MAX

LV_PCT(x)

LV_COORD_IS_PCT(x)

LV_COORD_GET_PCT(x)

Enums

enum **lv_align_t**

Alignments

Values:

enumerator **LV_ALIGN_DEFAULT**

enumerator **LV_ALIGN_TOP_LEFT**

enumerator **LV_ALIGN_TOP_MID**

enumerator **LV_ALIGN_TOP_RIGHT**

enumerator **LV_ALIGN_BOTTOM_LEFT**

enumerator **LV_ALIGN_BOTTOM_MID**

enumerator **LV_ALIGN_BOTTOM_RIGHT**

enumerator **LV_ALIGN_LEFT_MID**

enumerator **LV_ALIGN_RIGHT_MID**

enumerator **LV_ALIGN_CENTER**

enumerator **LV_ALIGN_OUT_TOP_LEFT**

enumerator **LV_ALIGN_OUT_TOP_MID**

enumerator **LV_ALIGN_OUT_TOP_RIGHT**

enumerator **LV_ALIGN_OUT_BOTTOM_LEFT**

enumerator **LV_ALIGN_OUT_BOTTOM_MID**

enumerator **LV_ALIGN_OUT_BOTTOM_RIGHT**

enumerator **LV_ALIGN_OUT_LEFT_TOP**

enumerator **LV_ALIGN_OUT_LEFT_MID**

enumerator **LV_ALIGN_OUT_LEFT_BOTTOM**

enumerator **LV_ALIGN_OUT_RIGHT_TOP**

enumerator **LV_ALIGN_OUT_RIGHT_MID**

enumerator **LV_ALIGN_OUT_RIGHT_BOTTOM**

enum **lv_dir_t**

Values:

enumerator **LV_DIR_NONE**

enumerator **LV_DIR_LEFT**

enumerator **LV_DIR_RIGHT**

enumerator **LV_DIR_TOP**

enumerator **LV_DIR_BOTTOM**

enumerator **LV_DIR_HOR**

enumerator **LV_DIR_VER**

enumerator **LV_DIR_ALL**

Functions

void **lv_area_set**(*lv_area_t* *area_p, int32_t x1, int32_t y1, int32_t x2, int32_t y2)

Initialize an area

Parametri

- **area_p** -- pointer to an area
- **x1** -- left coordinate of the area
- **y1** -- top coordinate of the area
- **x2** -- right coordinate of the area
- **y2** -- bottom coordinate of the area

static inline void **lv_area_copy**(*lv_area_t* *dest, const *lv_area_t* *src)

Copy an area

Parametri

- **dest** -- pointer to the destination area
- **src** -- pointer to the source area

int32_t **lv_area_get_width**(const *lv_area_t* *area_p)

Get the width of an area

Parametri

area_p -- pointer to an area

Ritorna

the width of the area (if x1 == x2 -> width = 1)

int32_t **lv_area_get_height**(const *lv_area_t* *area_p)

Get the height of an area

Parametri

area_p -- pointer to an area

Ritorna

the height of the area (if y1 == y2 -> height = 1)

void **lv_area_set_width**(*lv_area_t* *area_p, int32_t w)

Set the width of an area

Parametri

- **area_p** -- pointer to an area
- **w** -- the new width of the area (w == 1 makes x1 == x2)

void **lv_area_set_height**(*lv_area_t* *area_p, int32_t h)

Set the height of an area

Parametri

- **area_p** -- pointer to an area
- **h** -- the new height of the area (h == 1 makes y1 == y2)

uint32_t **lv_area_get_size**(const *lv_area_t* *area_p)

Return with area of an area (x * y)

Parametri

area_p -- pointer to an area

Ritorna

size of area

void **lv_area_increase**(*lv_area_t* *area, int32_t w_extra, int32_t h_extra)

void **lv_area_move**(*lv_area_t* *area, int32_t x_ofs, int32_t y_ofs)

void **lv_area_align**(const *lv_area_t* *base, *lv_area_t* *to_align, *lv_align_t* align, int32_t ofs_x, int32_t ofs_y)

Align an area to another

Parametri

- **base** -- an area where the other will be aligned
- **to_align** -- the area to align
- **align** -- LV_ALIGN_...
- **ofs_x** -- X offset
- **ofs_y** -- Y offset

void **lv_point_transform**(*lv_point_t* *point, int32_t angle, int32_t scale_x, int32_t scale_y, const *lv_point_t* *pivot, bool zoom_first)

Transform a point

Parametri

- **point** -- pointer to a point
- **angle** -- angle with 0.1 resolutions (123 means 12.3°)
- **scale_x** -- horizontal zoom, 256 means 100%
- **scale_y** -- vertical zoom, 256 means 100%
- **pivot** -- pointer to the pivot point of the transformation
- **zoom_first** -- true: zoom first and rotate after that; else: opposite order

void **lv_point_array_transform**(*lv_point_t* *points, size_t count, int32_t angle, int32_t scale_x, int32_t scale_y, const *lv_point_t* *pivot, bool zoom_first)

Transform an array of points

Parametri

- **points** -- pointer to an array of points
- **count** -- number of points in the array
- **angle** -- angle with 0.1 resolutions (123 means 12.3°)
- **scale_x** -- horizontal zoom, 256 means 100%
- **scale_y** -- vertical zoom, 256 means 100%
- **pivot** -- pointer to the pivot point of the transformation
- **zoom_first** -- true: zoom first and rotate after that; else: opposite order

lv_point_t **lv_point_from_precise**(const *lv_point_precise_t* *p)

lv_point_precise_t **lv_point_to_precise**(const *lv_point_t* *p)

void **lv_point_set**(*lv_point_t* *p, int32_t x, int32_t y)

void **lv_point_precise_set**(*lv_point_precise_t* *p, lv_value_precise_t x, lv_value_precise_t y)

void **lv_point_swap**(*lv_point_t* *p1, *lv_point_t* *p2)

void **lv_point_precise_swap**(*lv_point_precise_t* *p1, *lv_point_precise_t* *p2)

int32_t **lv_pct**(int32_t x)

Convert a percentage value to `int32_t`. Percentage values are stored in special range

Parametri

x -- the percentage (0..1000)

Ritorna

a coordinate that stores the percentage

int32_t **lv_pct_to_px**(int32_t v, int32_t base)

struct **lv_point_t**

#include <lv_area.h> Represents a point on the screen.

Public Members

int32_t **x**

int32_t **y**

struct **lv_point_precise_t**

Public Members

lv_value_precise_t **x**

lv_value_precise_t **y**

struct **lv_area_t**

#include <lv_area.h> Represents an area of the screen.

Public Members

int32_t **x1**

int32_t **y1**

int32_t **x2**

int32_t **y2**

14.18.5 lv_area_private.h

Functions

void **lv_area_set_pos**(*lv_area_t* *area_p, int32_t x, int32_t y)

Set the position of an area (width and height will be kept)

Parametri

- **area_p** -- pointer to an area
- **x** -- the new x coordinate of the area
- **y** -- the new y coordinate of the area

bool **lv_area_intersect**(*lv_area_t* *res_p, const *lv_area_t* *a1_p, const *lv_area_t* *a2_p)

Get the common parts of two areas

Parametri

- **res_p** -- pointer to an area, the result will be stored her
- **a1_p** -- pointer to the first area
- **a2_p** -- pointer to the second area

Ritorna

false: the two area has NO common parts, res_p is invalid

int8_t **lv_area_diff**(*lv_area_t* res_p[], const *lv_area_t* *a1_p, const *lv_area_t* *a2_p)

Get resulting sub areas after removing the common parts of two areas from the first area

Parametri

- **res_p** -- pointer to an array of areas with a count of 4, the resulting areas will be stored here
- **a1_p** -- pointer to the first area
- **a2_p** -- pointer to the second area

Ritorna

number of results (max 4) or -1 if no intersect

void **lv_area_join**(*lv_area_t* *a_res_p, const *lv_area_t* *a1_p, const *lv_area_t* *a2_p)

Join two areas into a third which involves the other two

Parametri

- **a_res_p** -- pointer to an area, the result will be stored here
- **a1_p** -- pointer to the first area
- **a2_p** -- pointer to the second area

bool **lv_area_is_point_on**(const *lv_area_t* *a_p, const *lv_point_t* *p_p, int32_t radius)

Check if a point is on an area

Parametri

- **a_p** -- pointer to an area
- **p_p** -- pointer to a point
- **radius** -- radius of area (e.g. for rounded rectangle)

Ritorna

false:the point is out of the area

bool **lv_area_is_on**(const *lv_area_t* *a1_p, const *lv_area_t* *a2_p)

Check if two area has common parts

Parametri

- **a1_p** -- pointer to an area.
- **a2_p** -- pointer to another area

Ritorna

false: a1_p and a2_p has no common parts

bool **lv_area_is_in**(const *lv_area_t* *ain_p, const *lv_area_t* *aholder_p, int32_t radius)

Check if an area is fully on another

Parametri

- **ain_p** -- pointer to an area which could be in 'aholder_p'
- **aholder_p** -- pointer to an area which could involve 'ain_p'
- **radius** -- radius of `aholder_p` (e.g. for rounded rectangle)

Ritorna

true: `ain_p` is fully inside `aholder_p`

bool **lv_area_is_out**(const *lv_area_t* *aout_p, const *lv_area_t* *aholder_p, int32_t radius)

Check if an area is fully out of another

Parametri

- **aout_p** -- pointer to an area which could be in 'aholder_p'
- **aholder_p** -- pointer to an area which could involve 'ain_p'
- **radius** -- radius of `aholder_p` (e.g. for rounded rectangle)

Ritorna

true: `aout_p` is fully outside `aholder_p`

bool **lv_area_is_equal**(const *lv_area_t* *a, const *lv_area_t* *b)

Check if 2 area is the same

Parametri

- **a** -- pointer to an area
- **b** -- pointer to another area

14.18.6 lv_array.h

Array. The elements are dynamically allocated by the 'lv_mem' module.

Defines

LV_ARRAY_DEFAULT_CAPACITY

LV_ARRAY_DEFAULT_SHRINK_RATIO

Functions

void **lv_array_init**(lv_array_t *array, uint32_t capacity, uint32_t element_size)

Init an array.

Parametri

- **array** -- pointer to an `lv_array_t` variable to initialize
- **capacity** -- the initial capacity of the array
- **element_size** -- the size of an element in bytes

void **lv_array_init_from_buf**(lv_array_t *array, void *buf, uint32_t capacity, uint32_t element_size)

Init an array from a buffer.

> **Nota**
>
> The buffer must be large enough to store `capacity` elements. The array will not release the buffer and reallocate it. The user must ensure that the buffer is valid during the lifetime of the array. And release the buffer when the array is no longer needed.

Parametri

- **array** -- pointer to an `lv_array_t` variable to initialize
- **buf** -- pointer to a buffer to use as the array's data
- **capacity** -- the initial capacity of the array
- **element_size** -- the size of an element in bytes

bool **lv_array_resize**(lv_array_t *array, uint32_t new_capacity)

Resize the array to the given capacity.

> **Nota**
>
> if the new capacity is smaller than the current size, the array will be truncated.

Parametri

- **array** -- pointer to an `lv_array_t` variable
- **new_capacity** -- the new capacity of the array

void **lv_array_deinit**(lv_array_t *array)

Deinit the array, and free the allocated memory

Parametri

array -- pointer to an `lv_array_t` variable to deinitialize

static inline uint32_t **lv_array_size**(const lv_array_t *array)

Return how many elements are stored in the array.

Parametri

array -- pointer to an `lv_array_t` variable

Ritorna

the number of elements stored in the array

static inline uint32_t **lv_array_capacity**(const lv_array_t *array)

Return the capacity of the array, i.e. how many elements can be stored.

Parametri

array -- pointer to an `lv_array_t` variable

Ritorna

the capacity of the array

static inline bool **lv_array_is_empty**(const lv_array_t *array)

Return if the array is empty

Parametri

array -- pointer to an `lv_array_t` variable

Ritorna

true: array is empty; false: array is not empty

static inline bool **lv_array_is_full**(const lv_array_t *array)

Return if the array is full

Parametri

array -- pointer to an `lv_array_t` variable

Ritorna

true: array is full; false: array is not full

void **lv_array_copy**(lv_array_t *target, const lv_array_t *source)

Copy an array to another.

> **Nota**
>
> this will create a new array with the same capacity and size as the source array.

Parametri

- **target** -- pointer to an `lv_array_t` variable to copy to
- **source** -- pointer to an `lv_array_t` variable to copy from

static inline void **lv_array_clear**(lv_array_t *array)

Remove all elements in array.

Parametri

array -- pointer to an `lv_array_t` variable

void **lv_array_shrink**(lv_array_t *array)

Shrink the memory capacity of array if necessary.

Parametri

array -- pointer to an `lv_array_t` variable

lv_result_t **lv_array_remove**(lv_array_t *array, uint32_t index)

Remove the element at the specified position in the array.

Parametri

- **array** -- pointer to an `lv_array_t` variable
- **index** -- the index of the element to remove

Ritorna

LV_RESULT_OK: success, otherwise: error

lv_result_t **lv_array_erase**(lv_array_t *array, uint32_t start, uint32_t end)

Remove from the array either a single element or a range of elements ([start, end)).

> **Nota**
>
> This effectively reduces the container size by the number of elements removed.

> **Nota**
>
> When start equals to end, the function has no effect.

Parametri

- **array** -- pointer to an `lv_array_t` variable
- **start** -- the index of the first element to be removed
- **end** -- the index of the first element that is not to be removed

Ritorna

LV_RESULT_OK: success, otherwise: error

lv_result_t **lv_array_concat**(lv_array_t *array, const lv_array_t *other)

Concatenate two arrays. Adds new elements to the end of the array.

> **Nota**
>
> The destination array is automatically expanded as necessary.

Parametri

- **array** -- pointer to an `lv_array_t` variable
- **other** -- pointer to the array to concatenate

Ritorna

LV_RESULT_OK: success, otherwise: error

lv_result_t **lv_array_push_back**(lv_array_t *array, const void *element)

Push back element. Adds a new element to the end of the array. If the array capacity is not enough for the new element, the array will be resized automatically.

> **Nota**
>
> If the element is NULL, it will be added as an empty element.

Parametri

- **array** -- pointer to an `lv_array_t` variable
- **element** -- pointer to the element to add. NULL to push an empty element.

Ritorna

LV_RESULT_OK: success, otherwise: error

lv_result_t **lv_array_assign**(lv_array_t *array, uint32_t index, const void *value)

Assigns one content to the array, replacing its current content.

Parametri

- **array** -- pointer to an `lv_array_t` variable
- **index** -- the index of the element to replace
- **value** -- pointer to the elements to add

Ritorna

true: success; false: error

void ***lv_array_at**(const lv_array_t *array, uint32_t index)

Returns a pointer to the element at position n in the array.

Parametri

- **array** -- pointer to an `lv_array_t` variable
- **index** -- the index of the element to return

Ritorna
a pointer to the requested element, NULL if `index` is out of range

static inline void *`lv_array_front`(const lv_array_t *array)

Returns a pointer to the first element in the array.

Parametri
`array` -- pointer to an `lv_array_t` variable

Ritorna
a pointer to the first element in the array

static inline void *`lv_array_back`(const lv_array_t *array)

Returns a pointer to the last element in the array.

Parametri
`array` -- pointer to an `lv_array_t` variable

struct `_lv_array_t`

#include <lv_array.h> Description of a array

Public Members

uint8_t *`data`

uint32_t `size`

uint32_t `capacity`

uint32_t `element_size`

bool `inner_alloc`

14.18.7 lv_assert.h

Defines

`LV_ASSERT`(expr)

`LV_ASSERT_MSG`(expr, msg)

`LV_ASSERT_FORMAT_MSG`(expr, format, ...)

`LV_ASSERT_NULL`(p)

`LV_ASSERT_MALLOC`(p)

`LV_ASSERT_MEM_INTEGRITY`()

Functions

void `lv_assert_handler`(void)

14.18.8 lv_async.h

Typedef

typedef void (*`lv_async_cb_t`)(void*)

Type for async callback.

Functions

lv_result_t **lv_async_call**(*lv_async_cb_t* async_xcb, void *user_data)

Call an asynchronous function the next time *lv_timer_handler()* is run. This function is likely to return **before** the call actually happens!

Parametri

- **async_xcb** -- a callback which is the task itself. (the 'x' in the argument name indicates that it's not a fully generic function because it not follows the `func_name(object, callback, ...)` convention)
- **user_data** -- custom parameter

lv_result_t **lv_async_call_cancel**(*lv_async_cb_t* async_xcb, void *user_data)

Cancel an asynchronous function call

Parametri

- **async_xcb** -- a callback which is the task itself.
- **user_data** -- custom parameter

14.18.9 lv_bidi.h

Defines

LV_BIDI_LRO

Special non printable strong characters. They can be inserted to texts to affect the run's direction

LV_BIDI_RLO

Enums

enum **lv_base_dir_t**

Values:

enumerator **LV_BASE_DIR_LTR**

enumerator **LV_BASE_DIR_RTL**

enumerator **LV_BASE_DIR_AUTO**

enumerator **LV_BASE_DIR_NEUTRAL**

enumerator **LV_BASE_DIR_WEAK**

Functions

void **lv_bidi_calculate_align**(*lv_text_align_t* *align, *lv_base_dir_t* *base_dir, const char *txt)

Get the real text alignment from the a text alignment, base direction and a text.

Parametri

- **align** -- LV_TEXT_ALIGN_..., write back the calculated align here (LV_TEXT_ALIGN_LEFT/RIGHT/CENTER)
- **base_dir** -- LV_BASE_DIR_..., write the calculated base dir here (LV_BASE_DIR_LTR/RTL)
- **txt** -- a text, used with LV_BASE_DIR_AUTO to determine the base direction

void **lv_bidi_set_custom_neutrals_static**(const char *neutrals)

Set custom neutrals string

Parametri

neutrals -- default " \t\n\r.,:;'\"`!?%/\\-=()[]{}<>@#&$|"

14.18.10 lv_bidi_private.h

Functions

void **lv_bidi_process**(const char *str_in, char *str_out, *lv_base_dir_t* base_dir)

Convert a text to get the characters in the correct visual order according to Unicode Bidirectional Algorithm

Parametri

- **str_in** -- the text to process
- **str_out** -- store the result here. Has the be `strlen(str_in)` length
- **base_dir** -- `LV_BASE_DIR_LTR` or `LV_BASE_DIR_RTL`

lv_base_dir_t **lv_bidi_detect_base_dir**(const char *txt)

Auto-detect the direction of a text based on the first strong character

Parametri

txt -- the text to process

Ritorna

`LV_BASE_DIR_LTR` or `LV_BASE_DIR_RTL`

uint16_t **lv_bidi_get_logical_pos**(const char *str_in, char **bidi_txt, uint32_t len, *lv_base_dir_t* base_dir, uint32_t visual_pos, bool *is_rtl)

Get the logical position of a character in a line

Parametri

- **str_in** -- the input string. Can be only one line.
- **bidi_txt** -- internally the text is bidi processed which buffer can be get here. If not required anymore has to freed with *`lv_free()`* Can be `NULL` is unused
- **len** -- length of the line in character count
- **base_dir** -- base direction of the text: `LV_BASE_DIR_LTR` or `LV_BASE_DIR_RTL`
- **visual_pos** -- the visual character position which logical position should be get
- **is_rtl** -- tell the char at `visual_pos` is RTL or LTR context

Ritorna

the logical character position

uint16_t **lv_bidi_get_visual_pos**(const char *str_in, char **bidi_txt, uint16_t len, *lv_base_dir_t* base_dir, uint32_t logical_pos, bool *is_rtl)

Get the visual position of a character in a line

Parametri

- **str_in** -- the input string. Can be only one line.
- **bidi_txt** -- internally the text is bidi processed which buffer can be get here. If not required anymore has to freed with *`lv_free()`* Can be `NULL` is unused
- **len** -- length of the line in character count
- **base_dir** -- base direction of the text: `LV_BASE_DIR_LTR` or `LV_BASE_DIR_RTL`
- **logical_pos** -- the logical character position which visual position should be get
- **is_rtl** -- tell the char at `logical_pos` is RTL or LTR context

Ritorna

the visual character position

void **lv_bidi_process_paragraph**(const char *str_in, char *str_out, uint32_t len, *lv_base_dir_t* base_dir, uint16_t *pos_conv_out, uint16_t pos_conv_len)

Bidi process a paragraph of text

Parametri

- **str_in** -- the string to process
- **str_out** -- store the result here

- **len** -- length of the text
- **base_dir** -- base dir of the text
- **pos_conv_out** -- an `uint16_t` array to store the related logical position of the character. Can be `NULL` is unused
- **pos_conv_len** -- length of `pos_conv_out` in element count

14.18.11 lv_circle_buf.h

Typedef

typedef bool (*`lv_circle_buf_fill_cb_t`)(void *buf, uint32_t buff_len, int32_t index, void *user_data)

Functions

lv_circle_buf_t *`lv_circle_buf_create`(uint32_t capacity, uint32_t element_size)

Create a circle buffer

Parametri

- **capacity** -- the maximum number of elements in the buffer
- **element_size** -- the size of an element in bytes

Ritorna

pointer to the created buffer

lv_circle_buf_t *`lv_circle_buf_create_from_buf`(void *buf, uint32_t capacity, uint32_t element_size)

Create a circle buffer from an existing buffer

Parametri

- **buf** -- pointer to a buffer
- **capacity** -- the maximum number of elements in the buffer
- **element_size** -- the size of an element in bytes

Ritorna

pointer to the created buffer

lv_circle_buf_t *`lv_circle_buf_create_from_array`(const lv_array_t *array)

Create a circle buffer from an existing array

Parametri

array -- pointer to an array

Ritorna

pointer to the created buffer

lv_result_t `lv_circle_buf_resize`(lv_circle_buf_t *circle_buf, uint32_t capacity)

Resize the buffer

Parametri

- **circle_buf** -- pointer to a buffer
- **capacity** -- the new capacity of the buffer

Ritorna

LV_RESULT_OK: the buffer is resized; LV_RESULT_INVALID: the buffer is not resized

void `lv_circle_buf_destroy`(lv_circle_buf_t *circle_buf)

Destroy a circle buffer

Parametri

circle_buf -- pointer to buffer

uint32_t `lv_circle_buf_size`(const lv_circle_buf_t *circle_buf)

Get the size of the buffer

Parametri

circle_buf -- pointer to buffer

Ritorna

the number of elements in the buffer

uint32_t **lv_circle_buf_capacity**(const lv_circle_buf_t *circle_buf)

Get the capacity of the buffer

Parametri

circle_buf -- pointer to buffer

Ritorna

the maximum number of elements in the buffer

uint32_t **lv_circle_buf_remain**(const lv_circle_buf_t *circle_buf)

Get the remaining space in the buffer

Parametri

circle_buf -- pointer to buffer

Ritorna

the number of elements that can be written to the buffer

bool **lv_circle_buf_is_empty**(const lv_circle_buf_t *circle_buf)

Check if the buffer is empty

Parametri

circle_buf -- pointer to buffer

Ritorna

true: the buffer is empty; false: the buffer is not empty

bool **lv_circle_buf_is_full**(const lv_circle_buf_t *circle_buf)

Check if the buffer is full

Parametri

circle_buf -- pointer to buffer

Ritorna

true: the buffer is full; false: the buffer is not full

void **lv_circle_buf_reset**(lv_circle_buf_t *circle_buf)

Reset the buffer

Parametri

circle_buf -- pointer to buffer

Ritorna

LV_RESULT_OK: the buffer is reset; LV_RESULT_INVALID: the buffer is not reset

void ***lv_circle_buf_head**(const lv_circle_buf_t *circle_buf)

Get the head of the buffer

Parametri

circle_buf -- pointer to buffer

Ritorna

pointer to the head of the buffer

void ***lv_circle_buf_tail**(const lv_circle_buf_t *circle_buf)

Get the tail of the buffer

Parametri

circle_buf -- pointer to buffer

Ritorna

pointer to the tail of the buffer

lv_result_t **lv_circle_buf_read**(lv_circle_buf_t *circle_buf, void *data)

Read a value

Parametri

- **circle_buf** -- pointer to buffer
- **data** -- pointer to a variable to store the read value

Ritorna

LV_RESULT_OK: the value is read; LV_RESULT_INVALID: the value is not read

lv_result_t **lv_circle_buf_write**(lv_circle_buf_t *circle_buf, const void *data)

Write a value

Parametri

- **circle_buf** -- pointer to buffer
- **data** -- pointer to the value to write

Ritorna

LV_RESULT_OK: the value is written; LV_RESULT_INVALID: the value is not written

uint32_t **lv_circle_buf_fill**(lv_circle_buf_t *circle_buf, uint32_t count, *lv_circle_buf_fill_cb_t* fill_cb, void *user_data)

Fill the buffer with values

Parametri

- **circle_buf** -- pointer to buffer
- **count** -- the number of values to fill
- **fill_cb** -- the callback function to fill the buffer
- **user_data** --

Ritorna

the number of values filled

lv_result_t **lv_circle_buf_skip**(lv_circle_buf_t *circle_buf)

Skip a value

Parametri

circle_buf -- pointer to buffer

Ritorna

LV_RESULT_OK: the value is skipped; LV_RESULT_INVALID: the value is not skipped

lv_result_t **lv_circle_buf_peek**(const lv_circle_buf_t *circle_buf, void *data)

Peek a value

Parametri

- **circle_buf** -- pointer to buffer
- **data** -- pointer to a variable to store the peeked value

Ritorna

LV_RESULT_OK: the value is peeked; LV_RESULT_INVALID: the value is not peeked

lv_result_t **lv_circle_buf_peek_at**(const lv_circle_buf_t *circle_buf, uint32_t index, void *data)

Peek a value at an index

Parametri

- **circle_buf** -- pointer to buffer
- **index** -- the index of the value to peek, if the index is greater than the size of the buffer, it will return looply.
- **data** -- pointer to a variable to store the peeked value

Ritorna

LV_RESULT_OK: the value is peeked; LV_RESULT_INVALID: the value is not peeked

14.18.12 lv_color.h

Defines

LV_COLOR_NATIVE_WITH_ALPHA_SIZE

LV_OPA_MIN

Fully transparent if opa <= LV_OPA_MIN

LV_OPA_MAX

Fully cover if opa >= LV_OPA_MAX

LV_COLOR_FORMAT_GET_BPP(cf)

Get the pixel size of a color format in bits, bpp

Vedi anche

lv_color_format_get_bpp

Parametri

- **cf** -- a color format (LV_COLOR_FORMAT_...)

Ritorna

the pixel size in bits

LV_COLOR_FORMAT_GET_SIZE(cf)

Get the pixel size of a color format in bytes

Vedi anche

lv_color_format_get_size

Parametri

- **cf** -- a color format (LV_COLOR_FORMAT_...)

Ritorna

the pixel size in bytes

LV_COLOR_FORMAT_IS_ALPHA_ONLY(cf)

LV_COLOR_FORMAT_IS_INDEXED(cf)

LV_COLOR_FORMAT_IS_YUV(cf)

LV_COLOR_INDEXED_PALETTE_SIZE(cf)

LV_COLOR_MAKE(r8, g8, b8)

LV_OPA_MIX2(a1, a2)

LV_OPA_MIX3(a1, a2, a3)

Enums

enum **_lv_opacity_level_t**

Opacity percentages.

Values:

enumerator **LV_OPA_TRANSP**

enumerator **LV_OPA_0**

enumerator **LV_OPA_10**

enumerator **LV_OPA_20**

enumerator **LV_OPA_30**

enumerator **LV_OPA_40**

enumerator **LV_OPA_50**

enumerator **LV_OPA_60**

enumerator **LV_OPA_70**

enumerator **LV_OPA_80**

enumerator **LV_OPA_90**

enumerator **LV_OPA_100**

enumerator **LV_OPA_COVER**

enum **lv_color_format_t**

Values:

enumerator **LV_COLOR_FORMAT_UNKNOWN**

enumerator **LV_COLOR_FORMAT_RAW**

enumerator **LV_COLOR_FORMAT_RAW_ALPHA**

enumerator **LV_COLOR_FORMAT_L8**

enumerator **LV_COLOR_FORMAT_I1**

enumerator **LV_COLOR_FORMAT_I2**

enumerator **LV_COLOR_FORMAT_I4**

enumerator **LV_COLOR_FORMAT_I8**

enumerator **LV_COLOR_FORMAT_A8**

enumerator **LV_COLOR_FORMAT_RGB565**

enumerator **LV_COLOR_FORMAT_ARGB8565**

Not supported by sw renderer yet.

enumerator **LV_COLOR_FORMAT_RGB565A8**

Color array followed by Alpha array

enumerator **LV_COLOR_FORMAT_AL88**

L8 with alpha >

enumerator **LV_COLOR_FORMAT_RGB565_SWAPPED**

enumerator **LV_COLOR_FORMAT_RGB888**

enumerator **LV_COLOR_FORMAT_ARGB8888**

enumerator **LV_COLOR_FORMAT_XRGB8888**

enumerator **LV_COLOR_FORMAT_ARGB8888_PREMULTIPLIED**

enumerator **LV_COLOR_FORMAT_A1**

enumerator **LV_COLOR_FORMAT_A2**

enumerator **LV_COLOR_FORMAT_A4**

enumerator **LV_COLOR_FORMAT_ARGB1555**

enumerator **LV_COLOR_FORMAT_ARGB4444**

enumerator **LV_COLOR_FORMAT_ARGB2222**

enumerator **LV_COLOR_FORMAT_YUV_START**

enumerator **LV_COLOR_FORMAT_I420**

enumerator **LV_COLOR_FORMAT_I422**

enumerator **LV_COLOR_FORMAT_I444**

enumerator **LV_COLOR_FORMAT_I400**

enumerator **LV_COLOR_FORMAT_NV21**

enumerator **LV_COLOR_FORMAT_NV12**

enumerator **LV_COLOR_FORMAT_YUY2**

enumerator **LV_COLOR_FORMAT_UYVY**

enumerator **LV_COLOR_FORMAT_YUV_END**

enumerator **LV_COLOR_FORMAT_PROPRIETARY_START**

enumerator **LV_COLOR_FORMAT_NEMA_TSC_START**

enumerator **LV_COLOR_FORMAT_NEMA_TSC4**

enumerator **LV_COLOR_FORMAT_NEMA_TSC6**

enumerator **LV_COLOR_FORMAT_NEMA_TSC6A**

enumerator **LV_COLOR_FORMAT_NEMA_TSC6AP**

enumerator **LV_COLOR_FORMAT_NEMA_TSC12**

enumerator **LV_COLOR_FORMAT_NEMA_TSC12A**

enumerator **LV_COLOR_FORMAT_NEMA_TSC_END**

enumerator **LV_COLOR_FORMAT_NATIVE**

enumerator **LV_COLOR_FORMAT_NATIVE_WITH_ALPHA**

Functions

uint8_t **lv_color_format_get_bpp**(*lv_color_format_t* cf)

Get the pixel size of a color format in bits, bpp

> **Vedi anche**
>
> *LV_COLOR_FORMAT_GET_BPP*

Parametri
cf -- a color format (LV_COLOR_FORMAT_...)

Ritorna
the pixel size in bits

uint8_t **lv_color_format_get_size**(*lv_color_format_t* cf)

Get the pixel size of a color format in bytes

> **Vedi anche**
>
> *LV_COLOR_FORMAT_GET_SIZE*

Parametri
cf -- a color format (LV_COLOR_FORMAT_...)

Ritorna
the pixel size in bytes

bool **lv_color_format_has_alpha**(*lv_color_format_t* src_cf)

Check if a color format has alpha channel or not

Parametri
src_cf -- a color format (LV_COLOR_FORMAT_...)

Ritorna
true: has alpha channel; false: doesn't have alpha channel

lv_color32_t **lv_color_to_32**(*lv_color_t* color, lv_opa_t opa)

Create an ARGB8888 color from RGB888 + alpha

Parametri

- **color** -- an RGB888 color

- **opa** -- the alpha value

Ritorna
the ARGB8888 color

uint32_t **lv_color_to_int**(*lv_color_t* c)

Convert an RGB888 color to an integer

Parametri
c -- an RGB888 color

Ritorna
c as an integer

bool **lv_color_eq**(*lv_color_t* c1, *lv_color_t* c2)

Check if two RGB888 color are equal

Parametri
- **c1** -- the first color
- **c2** -- the second color

Ritorna
true: equal

bool **lv_color32_eq**(*lv_color32_t* c1, *lv_color32_t* c2)

Check if two ARGB8888 color are equal

Parametri
- **c1** -- the first color
- **c2** -- the second color

Ritorna
true: equal

lv_color_t **lv_color_hex**(uint32_t c)

Create a color from 0x000000..0xffffff input

Parametri
c -- the hex input

Ritorna
the color

lv_color_t **lv_color_make**(uint8_t r, uint8_t g, uint8_t b)

Create an RGB888 color

Parametri
- **r** -- the red channel (0..255)
- **g** -- the green channel (0..255)
- **b** -- the blue channel (0..255)

Ritorna
the color

lv_color32_t **lv_color32_make**(uint8_t r, uint8_t g, uint8_t b, uint8_t a)

Create an ARGB8888 color

Parametri
- **r** -- the red channel (0..255)
- **g** -- the green channel (0..255)
- **b** -- the blue channel (0..255)
- **a** -- the alpha channel (0..255)

Ritorna
the color

lv_color_t **lv_color_hex3**(uint32_t c)

Create a color from 0x000..0xfff input

Parametri
c -- the hex input (e.g. 0x123 will be 0x112233)

Ritorna
the color

uint16_t **lv_color_to_u16**(*lv_color_t* color)

Convert am RGB888 color to RGB565 stored in `uint16_t`

Parametri

color -- and RGB888 color

Ritorna

`color` as RGB565 on `uin16_t`

uint32_t **lv_color_to_u32**(*lv_color_t* color)

Convert am RGB888 color to XRGB8888 stored in `uint32_t`

Parametri

color -- and RGB888 color

Ritorna

`color` as XRGB8888 on `uin32_t` (the alpha channel is always set to 0xFF)

uint16_t **lv_color_16_16_mix**(uint16_t c1, uint16_t c2, uint8_t mix)

Mix two RGB565 colors

Parametri

- **c1** -- the first color (typically the foreground color)
- **c2** -- the second color (typically the background color)
- **mix** -- 0..255, or LV_OPA_0/10/20...

Ritorna

mix == 0: c2 mix == 255: c1 mix == 128: 0.5 x c1 + 0.5 x c2

lv_color_t **lv_color_lighten**(*lv_color_t* c, lv_opa_t lvl)

Mix white to a color

Parametri

- **c** -- the base color
- **lvl** -- the intensity of white (0: no change, 255: fully white)

Ritorna

the mixed color

lv_color_t **lv_color_darken**(*lv_color_t* c, lv_opa_t lvl)

Mix black to a color

Parametri

- **c** -- the base color
- **lvl** -- the intensity of black (0: no change, 255: fully black)

Ritorna

the mixed color

lv_color_t **lv_color_hsv_to_rgb**(uint16_t h, uint8_t s, uint8_t v)

Convert a HSV color to RGB

Parametri

- **h** -- hue [0..359]
- **s** -- saturation [0..100]
- **v** -- value [0..100]

Ritorna

the given RGB color in RGB (with LV_COLOR_DEPTH depth)

lv_color_hsv_t **lv_color_rgb_to_hsv**(uint8_t r8, uint8_t g8, uint8_t b8)

Convert a 32-bit RGB color to HSV

Parametri

- **r8** -- 8-bit red
- **g8** -- 8-bit green
- **b8** -- 8-bit blue

Ritorna

the given RGB color in HSV

lv_color_hsv_t **lv_color_to_hsv**(*lv_color_t* color)

Convert a color to HSV

Parametri
color -- color
Ritorna
the given color in HSV

lv_color_t **lv_color_white**(void)

A helper for white color
Ritorna
a white color

lv_color_t **lv_color_black**(void)

A helper for black color
Ritorna
a black color

void **lv_color_premultiply**(*lv_color32_t* *c)

void **lv_color16_premultiply**(*lv_color16_t* *c, lv_opa_t a)

uint8_t **lv_color_luminance**(*lv_color_t* c)

Get the luminance of a color: luminance = 0.3 R + 0.59 G + 0.11 B
Parametri
c -- a color
Ritorna
the brightness [0..255]

uint8_t **lv_color16_luminance**(const *lv_color16_t* c)

Get the luminance of a color16: luminance = 0.3 R + 0.59 G + 0.11 B
Parametri
c -- a color
Ritorna
the brightness [0..255]

uint8_t **lv_color24_luminance**(const uint8_t *c)

Get the luminance of a color24: luminance = 0.3 R + 0.59 G + 0.11 B
Parametri
c -- a color
Ritorna
the brightness [0..255]

uint8_t **lv_color32_luminance**(*lv_color32_t* c)

Get the luminance of a color32: luminance = 0.3 R + 0.59 G + 0.11 B
Parametri
c -- a color
Ritorna
the brightness [0..255]

static inline uint16_t **lv_color_swap_16**(uint16_t c)

Swap the endianness of an rgb565 color
Parametri
c -- a color
Ritorna
the swapped color

Variables

const lv_color_filter_dsc_t **lv_color_filter_shade**

struct **lv_color_t**

Public Members

uint8_t **blue**

uint8_t **green**

uint8_t **red**

struct **lv_color16_t**

Public Members

uint16_t **blue**

uint16_t **green**

uint16_t **red**

struct **lv_color32_t**

Public Members

uint8_t **blue**

uint8_t **green**

uint8_t **red**

uint8_t **alpha**

struct **lv_color_hsv_t**

Public Members

uint16_t **h**

uint8_t **s**

uint8_t **v**

struct **lv_color16a_t**

Public Members

uint8_t **lumi**

uint8_t **alpha**

14.18.13 lv_color_op.h

Typedef

typedef *lv_color_t* (*`lv_color_filter_cb_t`)(const struct *_lv_color_filter_dsc_t**, *lv_color_t*, lv_opa_t)

Functions

lv_color_t `lv_color_mix`(*lv_color_t* c1, *lv_color_t* c2, uint8_t mix)

Mix two colors with a given ratio.

Parametri

- `c1` -- the first color to mix (usually the foreground)
- `c2` -- the second color to mix (usually the background)
- `mix` -- The ratio of the colors. 0: full `c2`, 255: full `c1`, 127: half `c1` and half `c2`

Ritorna

the mixed color

lv_color32_t `lv_color_mix32`(*lv_color32_t* fg, *lv_color32_t* bg)

> **Nota**
>
> Use bg.alpha in the return value

> **Nota**
>
> Use fg.alpha as mix ratio

Parametri

- `fg` --
- `bg` --

Ritorna

lv_color32_t `lv_color_mix32_premultiplied`(*lv_color32_t* fg, *lv_color32_t* bg)

Blends two premultiplied ARGB8888 colors while maintaining correct alpha compositing.

This function correctly blends the foreground (fg) and background (bg) colors, ensuring that the output remains in a premultiplied alpha format.

> **Nota**
>
> If the foreground is fully opaque, it is returned as is.

> **Nota**
>
> If the foreground is fully transparent, the background is returned.

Parametri

- `fg` -- The foreground color in premultiplied ARGB8888 format.
- `bg` -- The background color in premultiplied ARGB8888 format.

Ritorna

The resulting blended color in premultiplied ARGB8888 format.

uint8_t `lv_color_brightness`(*lv_color_t* c)

Get the brightness of a color

Parametri
c -- a color
Ritorna
brightness in range [0..255]

void **lv_color_filter_dsc_init**(lv_color_filter_dsc_t *dsc, *lv_color_filter_cb_t* cb)

lv_color32_t **lv_color_over32**(*lv_color32_t* fg, *lv_color32_t* bg)

Blend two colors that have not been pre-multiplied using their alpha values

Parametri
- **fg** -- the foreground color
- **bg** -- the background color

Ritorna
result color

struct **_lv_color_filter_dsc_t**

Public Members

lv_color_filter_cb_t **filter_cb**

void ***user_data**

14.18.14 lv_color_op_private.h

14.18.15 lv_event.h

Typedef

typedef void (***lv_event_cb_t**)(lv_event_t *e)

Enums

enum **lv_event_code_t**

Type of event being sent to Widget

Values:

enumerator **LV_EVENT_ALL**

enumerator **LV_EVENT_PRESSED**

Input device events Widget has been pressed

enumerator **LV_EVENT_PRESSING**

Widget is being pressed (sent continuously while pressing)

enumerator **LV_EVENT_PRESS_LOST**

Widget is still being pressed but slid cursor/finger off Widget

enumerator **LV_EVENT_SHORT_CLICKED**

Widget was pressed for a short period of time, then released. Not sent if scrolled.

enumerator **LV_EVENT_SINGLE_CLICKED**

Sent for first short click within a small distance and short time

enumerator **LV_EVENT_DOUBLE_CLICKED**

Sent for second short click within small distance and short time

enumerator **LV_EVENT_TRIPLE_CLICKED**

Sent for third short click within small distance and short time

enumerator **LV_EVENT_LONG_PRESSED**

Object has been pressed for at least `long_press_time`. Not sent if scrolled.

enumerator **LV_EVENT_LONG_PRESSED_REPEAT**

Sent after `long_press_time` in every `long_press_repeat_time` ms. Not sent if scrolled.

enumerator **LV_EVENT_CLICKED**

Sent on release if not scrolled (regardless to long press)

enumerator **LV_EVENT_RELEASED**

Sent in every cases when Widget has been released

enumerator **LV_EVENT_SCROLL_BEGIN**

Scrolling begins. The event parameter is a pointer to the animation of the scroll. Can be modified

enumerator **LV_EVENT_SCROLL_THROW_BEGIN**

enumerator **LV_EVENT_SCROLL_END**

Scrolling ends

enumerator **LV_EVENT_SCROLL**

Scrolling

enumerator **LV_EVENT_GESTURE**

A gesture is detected. Get gesture with `lv_indev_get_gesture_dir(lv_indev_active());`

enumerator **LV_EVENT_KEY**

A key is sent to Widget. Get key with `lv_indev_get_key(lv_indev_active());`

enumerator **LV_EVENT_ROTARY**

An encoder or wheel was rotated. Get rotation count with `lv_event_get_rotary_diff(e);`

enumerator **LV_EVENT_FOCUSED**

Widget received focus

enumerator **LV_EVENT_DEFOCUSED**

Widget's focus has been lost

enumerator **LV_EVENT_LEAVE**

Widget's focus has been lost but is still selected

enumerator **LV_EVENT_HIT_TEST**

Perform advanced hit-testing

enumerator **LV_EVENT_INDEV_RESET**

Indev has been reset

enumerator **LV_EVENT_HOVER_OVER**

Indev hover over object

enumerator **LV_EVENT_HOVER_LEAVE**

Indev hover leave object

enumerator **LV_EVENT_COVER_CHECK**

Drawing events Check if Widget fully covers an area. The event parameter is `lv_cover_check_info_t *`.

enumerator **LV_EVENT_REFR_EXT_DRAW_SIZE**

Get required extra draw area around Widget (e.g. for shadow). The event parameter is `int32_t *` to store the size.

enumerator **LV_EVENT_DRAW_MAIN_BEGIN**

Starting the main drawing phase

enumerator **LV_EVENT_DRAW_MAIN**

Perform the main drawing

enumerator **LV_EVENT_DRAW_MAIN_END**

Finishing the main drawing phase

enumerator **LV_EVENT_DRAW_POST_BEGIN**

Starting the post draw phase (when all children are drawn)

enumerator **LV_EVENT_DRAW_POST**

Perform the post draw phase (when all children are drawn)

enumerator **LV_EVENT_DRAW_POST_END**

Finishing the post draw phase (when all children are drawn)

enumerator **LV_EVENT_DRAW_TASK_ADDED**

Adding a draw task. The `LV_OBJ_FLAG_SEND_DRAW_TASK_EVENTS` flag needs to be set

enumerator **LV_EVENT_VALUE_CHANGED**

Special events Widget's value has changed (i.e. slider moved)

enumerator **LV_EVENT_INSERT**

Text has been inserted into Widget. The event data is `char *` being inserted.

enumerator **LV_EVENT_REFRESH**

Notify Widget to refresh something on it (for user)

enumerator **LV_EVENT_READY**

A process has finished

enumerator **LV_EVENT_CANCEL**

A process has been cancelled

enumerator **LV_EVENT_CREATE**

Other events Object is being created

enumerator **LV_EVENT_DELETE**

Object is being deleted

enumerator **LV_EVENT_CHILD_CHANGED**

Child was removed, added, or its size, position were changed

enumerator **LV_EVENT_CHILD_CREATED**

Child was created, always bubbles up to all parents

enumerator **LV_EVENT_CHILD_DELETED**

Child was deleted, always bubbles up to all parents

enumerator **LV_EVENT_SCREEN_UNLOAD_START**

A screen unload started, fired immediately when scr_load is called

enumerator **LV_EVENT_SCREEN_LOAD_START**

A screen load started, fired when the screen change delay is expired

enumerator **LV_EVENT_SCREEN_LOADED**

A screen was loaded

enumerator **LV_EVENT_SCREEN_UNLOADED**

A screen was unloaded

enumerator **LV_EVENT_SIZE_CHANGED**

Object coordinates/size have changed

enumerator **LV_EVENT_STYLE_CHANGED**

Object's style has changed

enumerator **LV_EVENT_LAYOUT_CHANGED**

A child's position position has changed due to a layout recalculation

enumerator **LV_EVENT_GET_SELF_SIZE**

Get internal size of a widget

enumerator **LV_EVENT_INVALIDATE_AREA**

Events of optional LVGL components An area is invalidated (marked for redraw). `lv_event_get_param(e)` returns a pointer to an *lv_area_t* object with the coordinates of the area to be invalidated. The area can be freely modified if needed to adapt it a special requirement of the display. Usually needed with monochrome displays to invalidate `N x 8` rows or columns in one pass.

enumerator **LV_EVENT_RESOLUTION_CHANGED**

Sent when the resolution changes due to *lv_display_set_resolution()* or *lv_display_set_rotation()*.

enumerator **LV_EVENT_COLOR_FORMAT_CHANGED**

Sent as a result of any call to *lv_display_set_color_format()*.

enumerator **LV_EVENT_REFR_REQUEST**

Sent when something happened that requires redraw.

enumerator **LV_EVENT_REFR_START**

Sent before a refreshing cycle starts. Sent even if there is nothing to redraw.

enumerator **LV_EVENT_REFR_READY**

Sent when refreshing has been completed (after rendering and calling flush callback). Sent even if no redraw happened.

enumerator **LV_EVENT_RENDER_START**

Sent just before rendering begins.

enumerator **LV_EVENT_RENDER_READY**

Sent after rendering has been completed (before calling flush callback)

enumerator **LV_EVENT_FLUSH_START**

Sent before flush callback is called.

enumerator **LV_EVENT_FLUSH_FINISH**

Sent after flush callback call has returned.

enumerator **LV_EVENT_FLUSH_WAIT_START**

Sent before flush wait callback is called.

enumerator **LV_EVENT_FLUSH_WAIT_FINISH**

Sent after flush wait callback call has returned.

enumerator **LV_EVENT_VSYNC**

enumerator **LV_EVENT_VSYNC_REQUEST**

enumerator **LV_EVENT_LAST**

enumerator **LV_EVENT_PREPROCESS**

Number of default events

enumerator **LV_EVENT_MARKED_DELETING**

This is a flag that can be set with an event so it's processed before the class default event processing

Functions

lv_result_t **lv_event_send**(*lv_event_list_t* *list, lv_event_t *e, bool preprocess)

Event callback. Events are used to notify the user of some action being taken on Widget. For details, see ::lv_event_t.

lv_event_dsc_t ***lv_event_add**(*lv_event_list_t* *list, *lv_event_cb_t* cb, *lv_event_code_t* filter, void *user_data)

bool **lv_event_remove_dsc**(*lv_event_list_t* *list, lv_event_dsc_t *dsc)

uint32_t **lv_event_get_count**(*lv_event_list_t* *list)

lv_event_dsc_t ***lv_event_get_dsc**(*lv_event_list_t* *list, uint32_t index)

lv_event_cb_t **lv_event_dsc_get_cb**(lv_event_dsc_t *dsc)

void ***lv_event_dsc_get_user_data**(lv_event_dsc_t *dsc)

bool **lv_event_remove**(*lv_event_list_t* *list, uint32_t index)

void **lv_event_remove_all**(*lv_event_list_t* *list)

void ***lv_event_get_target**(lv_event_t *e)

Get Widget originally targeted by the event. It's the same even if event was bubbled.

Parametri

e -- pointer to the event descriptor

Ritorna

the target of the event_code

void ***lv_event_get_current_target**(lv_event_t *e)

Get current target of the event. It's the Widget for which the event handler being called. If the event is not bubbled it's the same as "normal" target.

Parametri

e -- pointer to the event descriptor

Ritorna

pointer to the current target of the event_code

lv_event_code_t **lv_event_get_code**(lv_event_t *e)

Get event code of an event.

Parametri

e -- pointer to the event descriptor

Ritorna

the event code. (E.g. `LV_EVENT_CLICKED`, `LV_EVENT_FOCUSED`, etc)

void ***lv_event_get_param**(lv_event_t *e)

Get parameter passed when event was sent.

Parametri

e -- pointer to the event descriptor

Ritorna

pointer to the parameter

void ***lv_event_get_user_data**(lv_event_t *e)

Get user_data passed when event was registered on Widget.

Parametri
e -- pointer to the event descriptor
Ritorna
pointer to the user_data

void **lv_event_stop_bubbling**(lv_event_t *e)

Stop event from bubbling. This is only valid when called in the middle of an event processing chain.
Parametri
e -- pointer to the event descriptor

void **lv_event_stop_trickling**(lv_event_t *e)

Stop event from trickling down to children. This is only valid when called in the middle of an event processing chain.
Parametri
e -- pointer to the event descriptor

void **lv_event_stop_processing**(lv_event_t *e)

Stop processing this event. This is only valid when called in the middle of an event processing chain.
Parametri
e -- pointer to the event descriptor

uint32_t **lv_event_register_id**(void)

Register a new, custom event ID. It can be used the same way as e.g. `LV_EVENT_CLICKED` to send custom events

Example:

```
uint32_t LV_EVENT_MINE = 0;
...
e = lv_event_register_id();
...
lv_obj_send_event(obj, LV_EVENT_MINE, &some_data);
```

Ritorna
the new event id

const char ***lv_event_code_get_name**(*lv_event_code_t* code)

Get the name of an event code.
Parametri
code -- the event code
Ritorna
the name of the event code as a string

struct **lv_event_list_t**

Public Members

lv_array_t **array**

uint8_t **is_traversing**

True: the list is being nested traversed

uint8_t **has_marked_deleting**

True: the list has marked deleting objects when some of events are marked as deleting

14.18.16 lv_event_private.h

Functions

void **lv_event_push**(lv_event_t *e)

void **lv_event_pop**(lv_event_t *e)

void **lv_event_mark_deleted**(void *target)

Nested events can be called and one of them might belong to an object that is being deleted. Mark this object's `event_temp_data` deleted to know that its `lv_obj_send_event` should return `LV_RESULT_INVALID`

Parametri

target -- pointer to an event target which was deleted

struct **_lv_event_dsc_t**

Public Members

lv_event_cb_t **cb**

void ***user_data**

uint32_t **filter**

struct **_lv_event_t**

Public Members

void ***current_target**

void ***original_target**

lv_event_code_t **code**

void ***user_data**

void ***param**

lv_event_t ***prev**

uint8_t **deleted**

uint8_t **stop_processing**

uint8_t **stop_bubbling**

uint8_t **stop_trickling**

14.18.17 lv_fs.h

Defines

LV_FS_MAX_FN_LENGTH

LV_FS_MAX_PATH_LENGTH

LV_FS_CACHE_FROM_BUFFER

Typedef

typedef struct *_lv_fs_drv_t* **lv_fs_drv_t**

Enums

enum **lv_fs_res_t**

Errors in the file system module.

Values:

enumerator **LV_FS_RES_OK**

enumerator **LV_FS_RES_HW_ERR**

enumerator **LV_FS_RES_FS_ERR**

enumerator **LV_FS_RES_NOT_EX**

enumerator **LV_FS_RES_FULL**

enumerator **LV_FS_RES_LOCKED**

enumerator **LV_FS_RES_DENIED**

enumerator **LV_FS_RES_BUSY**

enumerator **LV_FS_RES_TOUT**

enumerator **LV_FS_RES_NOT_IMP**

enumerator **LV_FS_RES_OUT_OF_MEM**

enumerator **LV_FS_RES_INV_PARAM**

enumerator **LV_FS_RES_UNKNOWN**

enum **lv_fs_mode_t**

File open mode.

Values:

enumerator **LV_FS_MODE_WR**

enumerator **LV_FS_MODE_RD**

enum **lv_fs_whence_t**

Seek modes.

Values:

enumerator **LV_FS_SEEK_SET**

Set the position from absolutely (from the start of file)

enumerator **LV_FS_SEEK_CUR**

Set the position from the current position

enumerator **LV_FS_SEEK_END**

Set the position from the end of the file

Functions

void **lv_fs_drv_init**(*lv_fs_drv_t* *drv)

Initialize a file system driver with default values. It is used to ensure all fields have known values and not memory junk. After it you can set the fields.

Parametri

drv -- pointer to driver variable to initialize

void **lv_fs_drv_register**(*lv_fs_drv_t* *drv)

Add a new drive

Parametri

drv -- pointer to an lv_fs_drv_t structure which is inited with the corresponding function pointers. Only pointer is saved, so the driver should be static or dynamically allocated.

lv_fs_drv_t ***lv_fs_get_drv**(char letter)

Give a pointer to a driver from its letter

Parametri

letter -- the driver-identifier letter

Ritorna

pointer to a driver or NULL if not found

bool **lv_fs_is_ready**(char letter)

Test if a drive is ready or not. If the `ready` function was not initialized `true` will be returned.

Parametri

letter -- letter of the drive

Ritorna

true: drive is ready; false: drive is not ready

lv_fs_res_t **lv_fs_open**(*lv_fs_file_t* *file_p, const char *path, *lv_fs_mode_t* mode)

Open a file

Parametri

- **file_p** -- pointer to a *lv_fs_file_t* variable
- **path** -- path to the file beginning with the driver letter (e.g. S:/folder/file.txt)

- **mode** -- read: FS_MODE_RD, write: FS_MODE_WR, both: FS_MODE_RD | FS_MODE_WR

Ritorna
LV_FS_RES_OK or any error from *lv_fs_res_t* enum

void **lv_fs_make_path_from_buffer**(lv_fs_path_ex_t *path, char letter, const void *buf, uint32_t size)

Make a path object for the memory-mapped file compatible with the file system interface

Parametri
- **path** -- path to a lv_fs_path_ex object
- **letter** -- the identifier letter of the driver. E.g. `LV_FS_MEMFS_LETTER`
- **buf** -- address of the memory buffer
- **size** -- size of the memory buffer in bytes

lv_fs_res_t **lv_fs_close**(*lv_fs_file_t* *file_p)

Close an already opened file

Parametri
file_p -- pointer to a *lv_fs_file_t* variable

Ritorna
LV_FS_RES_OK or any error from *lv_fs_res_t* enum

lv_fs_res_t **lv_fs_read**(*lv_fs_file_t* *file_p, void *buf, uint32_t btr, uint32_t *br)

Read from a file

Parametri
- **file_p** -- pointer to a *lv_fs_file_t* variable
- **buf** -- pointer to a buffer where the read bytes are stored
- **btr** -- Bytes To Read
- **br** -- the number of real read bytes (Bytes Read). NULL if unused.

Ritorna
LV_FS_RES_OK or any error from *lv_fs_res_t* enum

lv_fs_res_t **lv_fs_write**(*lv_fs_file_t* *file_p, const void *buf, uint32_t btw, uint32_t *bw)

Write into a file

Parametri
- **file_p** -- pointer to a *lv_fs_file_t* variable
- **buf** -- pointer to a buffer with the bytes to write
- **btw** -- Bytes To Write
- **bw** -- the number of real written bytes (Bytes Written). NULL if unused.

Ritorna
LV_FS_RES_OK or any error from *lv_fs_res_t* enum

lv_fs_res_t **lv_fs_seek**(*lv_fs_file_t* *file_p, uint32_t pos, *lv_fs_whence_t* whence)

Set the position of the 'cursor' (read write pointer) in a file

Parametri
- **file_p** -- pointer to a *lv_fs_file_t* variable
- **pos** -- the new position expressed in bytes index (0: start of file)
- **whence** -- tells from where to set position. See *lv_fs_whence_t*

Ritorna
LV_FS_RES_OK or any error from *lv_fs_res_t* enum

lv_fs_res_t **lv_fs_tell**(*lv_fs_file_t* *file_p, uint32_t *pos)

Give the position of the read write pointer

Parametri
- **file_p** -- pointer to a *lv_fs_file_t* variable
- **pos** -- pointer to store the position of the read write pointer

Ritorna
LV_FS_RES_OK or any error from 'fs_res_t'

lv_fs_res_t **lv_fs_dir_open**(*lv_fs_dir_t* *rddir_p, const char *path)

Initialize a 'fs_dir_t' variable for directory reading

Parametri

- **rddir_p** -- pointer to a '*lv_fs_dir_t*' variable
- **path** -- path to a directory

Ritorna
LV_FS_RES_OK or any error from *lv_fs_res_t* enum

lv_fs_res_t **lv_fs_dir_read**(*lv_fs_dir_t* *rddir_p, char *fn, uint32_t fn_len)

Read the next filename form a directory. The name of the directories will begin with '/'

Parametri
- **rddir_p** -- pointer to an initialized 'fs_dir_t' variable
- **fn** -- pointer to a buffer to store the filename
- **fn_len** -- length of the buffer to store the filename

Ritorna
LV_FS_RES_OK or any error from *lv_fs_res_t* enum

lv_fs_res_t **lv_fs_dir_close**(*lv_fs_dir_t* *rddir_p)

Close the directory reading

Parametri
rddir_p -- pointer to an initialized 'fs_dir_t' variable

Ritorna
LV_FS_RES_OK or any error from *lv_fs_res_t* enum

char ***lv_fs_get_letters**(char *buf)

Fill a buffer with the letters of existing drivers

Parametri
buf -- buffer to store the letters ('\0' added after the last letter)

Ritorna
the buffer

const char ***lv_fs_get_ext**(const char *fn)

Return with the extension of the filename

Parametri
fn -- string with a filename

Ritorna
pointer to the beginning extension or empty string if no extension

char ***lv_fs_up**(char *path)

Step up one level

Parametri
path -- pointer to a file name

Ritorna
the truncated file name

const char ***lv_fs_get_last**(const char *path)

Get the last element of a path (e.g. U:/folder/file -> file)

Parametri
path -- pointer to a file name

Ritorna
pointer to the beginning of the last element in the path

struct **_lv_fs_drv_t**

Public Members

char **letter**

uint32_t **cache_size**

bool (*`ready_cb`)(*lv_fs_drv_t* *drv)

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

lv_fs_res_t (*`close_cb`)(*lv_fs_drv_t* *drv, void *file_p)

lv_fs_res_t (*`read_cb`)(*lv_fs_drv_t* *drv, void *file_p, void *buf, uint32_t btr, uint32_t *br)

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

lv_fs_res_t (*`seek_cb`)(*lv_fs_drv_t* *drv, void *file_p, uint32_t pos, *lv_fs_whence_t* whence)

lv_fs_res_t (*`tell_cb`)(*lv_fs_drv_t* *drv, void *file_p, uint32_t *pos_p)

void *(*`dir_open_cb`)(*lv_fs_drv_t* *drv, const char *path)

lv_fs_res_t (*`dir_read_cb`)(*lv_fs_drv_t* *drv, void *rddir_p, char *fn, uint32_t fn_len)

lv_fs_res_t (*`dir_close_cb`)(*lv_fs_drv_t* *drv, void *rddir_p)

void *`user_data`

Custom file user data

struct `lv_fs_file_t`

Public Members

void *`file_d`

lv_fs_drv_t *`drv`

lv_fs_file_cache_t *`cache`

struct `lv_fs_dir_t`

Public Members

void *`dir_d`

lv_fs_drv_t *`drv`

14.18.18 lv_fs_private.h

Functions

void `lv_fs_init`(void)

Initialize the File system interface

void **lv_fs_deinit**(void)

Deinitialize the File system interface

struct **_lv_fs_file_cache_t**

Public Members

uint32_t **start**

uint32_t **end**

uint32_t **file_position**

void ***buffer**

struct **_lv_fs_path_ex_t**

#include <lv_fs_private.h> Extended path object to specify buffer for memory-mapped files

Public Members

char **path**[4]

This is needed to make it compatible with a normal path

const void ***buffer**

uint32_t **size**

14.18.19 lv_grad.h

Enums

enum **lv_grad_dir_t**

The direction of the gradient.

Values:

enumerator **LV_GRAD_DIR_NONE**

No gradient (the `grad_color` property is ignored)

enumerator **LV_GRAD_DIR_VER**

Simple vertical (top to bottom) gradient

enumerator **LV_GRAD_DIR_HOR**

Simple horizontal (left to right) gradient

enumerator **LV_GRAD_DIR_LINEAR**

Linear gradient defined by start and end points. Can be at any angle.

enumerator **LV_GRAD_DIR_RADIAL**

Radial gradient defined by start and end circles

enumerator **LV_GRAD_DIR_CONICAL**

Conical gradient defined by center point, start and end angles

enum **lv_grad_extend_t**

Gradient behavior outside the defined range.

Values:

enumerator **LV_GRAD_EXTEND_PAD**

Repeat the same color

enumerator **LV_GRAD_EXTEND_REPEAT**

Repeat the pattern

enumerator **LV_GRAD_EXTEND_REFLECT**

Repeat the pattern mirrored

Functions

void **lv_grad_init_stops**(*lv_grad_dsc_t* *grad, const *lv_color_t* colors[], const lv_opa_t opa[], const uint8_t fracs[], int num_stops)

Initialize gradient color map from a table

Parametri

- **grad** -- pointer to a gradient descriptor
- **colors** -- color array
- **fracs** -- position array (0..255): if NULL, then colors are distributed evenly
- **opa** -- opacity array: if NULL, then LV_OPA_COVER is assumed
- **num_stops** -- number of gradient stops (1..LV_GRADIENT_MAX_STOPS)

void **lv_grad_horizontal_init**(*lv_grad_dsc_t* *dsc)

Helper function to initialize a horizontal gradient.

Parametri

dsc -- gradient descriptor

void **lv_grad_vertical_init**(*lv_grad_dsc_t* *dsc)

Helper function to initialize a vertical gradient.

Parametri

dsc -- gradient descriptor

void **lv_grad_linear_init**(*lv_grad_dsc_t* *dsc, int32_t from_x, int32_t from_y, int32_t to_x, int32_t to_y, *lv_grad_extend_t* extend)

Helper function to initialize linear gradient

Parametri

- **dsc** -- gradient descriptor
- **from_x** -- start x position: can be a coordinate or an *lv_pct()* value predefined constants LV_GRAD_LEFT, LV_GRAD_RIGHT, LV_GRAD_TOP, LV_GRAD_BOTTOM, LV_GRAD_CENTER can be used as well
- **from_y** -- start y position
- **to_x** -- end x position
- **to_y** -- end y position
- **extend** -- one of LV_GRAD_EXTEND_PAD, LV_GRAD_EXTEND_REPEAT or LV_GRAD_EXTEND_REFLECT

void **lv_grad_radial_init**(*lv_grad_dsc_t* *dsc, int32_t center_x, int32_t center_y, int32_t to_x, int32_t to_y, *lv_grad_extend_t* extend)

Helper function to initialize radial gradient

Parametri

- **dsc** -- gradient descriptor
- **center_x** -- center x position: can be a coordinate or an *lv_pct()* value predefined constants LV_GRAD_LEFT, LV_GRAD_RIGHT, LV_GRAD_TOP, LV_GRAD_BOTTOM, LV_GRAD_CENTER can be used as well
- **center_y** -- center y position
- **to_x** -- point on the end circle x position
- **to_y** -- point on the end circle y position
- **extend** -- one of LV_GRAD_EXTEND_PAD, LV_GRAD_EXTEND_REPEAT or LV_GRAD_EXTEND_REFLECT

void **lv_grad_radial_set_focal**(*lv_grad_dsc_t* *dsc, int32_t center_x, int32_t center_y, int32_t radius)

Set focal (starting) circle of a radial gradient

Parametri

- **dsc** -- gradient descriptor
- **center_x** -- center x position: can be a coordinate or an *lv_pct()* value predefined constants LV_GRAD_LEFT, LV_GRAD_RIGHT, LV_GRAD_TOP, LV_GRAD_BOTTOM, LV_GRAD_CENTER can be used as well
- **center_y** -- center y position
- **radius** -- radius of the starting circle (NOTE: this must be a scalar number, not percentage)

void **lv_grad_conical_init**(*lv_grad_dsc_t* *dsc, int32_t center_x, int32_t center_y, int32_t start_angle, int32_t end_angle, *lv_grad_extend_t* extend)

Helper function to initialize conical gradient

Parametri

- **dsc** -- gradient descriptor
- **center_x** -- center x position: can be a coordinate or an *lv_pct()* value predefined constants LV_GRAD_LEFT, LV_GRAD_RIGHT, LV_GRAD_TOP, LV_GRAD_BOTTOM, LV_GRAD_CENTER can be used as well
- **center_y** -- center y position
- **start_angle** -- start angle in degrees
- **end_angle** -- end angle in degrees
- **extend** -- one of LV_GRAD_EXTEND_PAD, LV_GRAD_EXTEND_REPEAT or LV_GRAD_EXTEND_REFLECT

struct **lv_grad_stop_t**

#include <lv_grad.h> A gradient stop definition. This matches a color and a position in a virtual 0-255 scale.

Public Members

lv_color_t **color**

The stop color

lv_opa_t **opa**

The opacity of the color

uint8_t **frac**

The stop position in 1/255 unit

struct **lv_grad_dsc_t**

#include <lv_grad.h> A descriptor of a gradient.

Public Members

lv_grad_stop_t **stops**[2]

A gradient stop array

uint8_t **stops_count**

The number of used stops in the array

lv_grad_dir_t **dir**

The gradient direction. Any of LV_GRAD_DIR_NONE, LV_GRAD_DIR_VER, LV_GRAD_DIR_HOR, LV_GRAD_TYPE_LINEAR, LV_GRAD_TYPE_RADIAL, LV_GRAD_TYPE_CONICAL

lv_grad_extend_t **extend**

Behaviour outside the defined range. LV_GRAD_EXTEND_NONE, LV_GRAD_EXTEND_PAD, LV_GRAD_EXTEND_REPEAT, LV_GRAD_EXTEND_REFLECT

lv_point_t **start**

Linear gradient vector start point

lv_point_t **end**

Linear gradient vector end point

Center of the ending circle in local coordinates

struct *lv_grad_dsc_t* **linear**

lv_point_t **focal**

Center of the focal (starting) circle in local coordinates

lv_point_t **focal_extent**

Point on the circle (can be the same as the center)

lv_point_t **end_extent**

Point on the circle determining the radius of the gradient

struct *lv_grad_dsc_t* **radial**

lv_point_t **center**

Conical gradient center point

int16_t **start_angle**

Start angle 0..3600

int16_t **end_angle**

End angle 0..3600

struct *lv_grad_dsc_t* **conical**

union *lv_grad_dsc_t* **params**

void ***state**

14.18.20 lv_iter.h

Typedef

typedef lv_result_t (***lv_iter_next_cb**)(void *instance, void *context, void *elem)

typedef void (***lv_iter_inspect_cb**)(void *elem)

Functions

lv_iter_t ***lv_iter_create**(void *instance, uint32_t elem_size, uint32_t context_size, *lv_iter_next_cb* next_cb)

Create an iterator based on an instance, and then the next element of the iterator can be obtained through lv_iter_next, In order to obtain the next operation in a unified and abstract way.

Parametri

- **instance** -- The instance to be iterated
- **elem_size** -- The size of the element to be iterated in bytes
- **context_size** -- The size of the context to be passed to the next_cb in bytes
- **next_cb** -- The callback function to get the next element

Ritorna

The iterator object

void ***lv_iter_get_context**(const lv_iter_t *iter)

Get the context of the iterator. You can use it to store some temporary variables associated with current iterator..

Parametri

iter -- `lv_iter_t` object create before

Ritorna

the iter context

void **lv_iter_destroy**(lv_iter_t *iter)

Destroy the iterator object, and release the context. Other resources allocated by the user are not released. The user needs to release it by itself.

Parametri

iter -- `lv_iter_t` object create before

lv_result_t **lv_iter_next**(lv_iter_t *iter, void *elem)

Get the next element of the iterator.

Parametri

- **iter** -- `lv_iter_t` object create before
- **elem** -- The pointer to store the next element

Ritorna

LV_RESULT_OK: Get the next element successfully LV_RESULT_INVALID: The next element is invalid

void **lv_iter_make_peekable**(lv_iter_t *iter, uint32_t capacity)

Make the iterator peekable, which means that the user can peek the next element without advancing the iterator.

Parametri

- **iter** -- `lv_iter_t` object create before
- **capacity** -- The capacity of the peek buffer

lv_result_t **lv_iter_peek**(lv_iter_t *iter, void *elem)

Peek the next element of the iterator without advancing the iterator.

Parametri

- **iter** -- `lv_iter_t` object create before

- **elem** -- The pointer to store the next element

Ritorna

LV_RESULT_OK: Peek the next element successfully LV_RESULT_INVALID: The next element is invalid

lv_result_t **lv_iter_peek_advance**(lv_iter_t *iter)

Only advance the iterator without getting the next element.

Parametri

iter -- `lv_iter_t` object create before

Ritorna

LV_RESULT_OK: Peek the next element successfully LV_RESULT_INVALID: The next element is invalid

lv_result_t **lv_iter_peek_reset**(lv_iter_t *iter)

Reset the peek cursor to the `next` cursor.

Parametri

iter -- `lv_iter_t` object create before

Ritorna

LV_RESULT_OK: Reset the peek buffer successfully LV_RESULT_INVALID: The peek buffer is invalid

void **lv_iter_inspect**(lv_iter_t *iter, *lv_iter_inspect_cb* inspect_cb)

Inspect the element of the iterator. The callback function will be called for each element of the iterator.

Parametri

- **iter** -- `lv_iter_t` object create before
- **inspect_cb** -- The callback function to inspect the element

14.18.21 lv_ll.h

Handle linked lists. The nodes are dynamically allocated by the 'lv_mem' module.

Defines

LV_LL_READ(list, i)

LV_LL_READ_BACK(list, i)

Typedef

typedef uint8_t **lv_ll_node_t**

Dummy type to make handling easier

Functions

void **lv_ll_init**(*lv_ll_t* *ll_p, uint32_t node_size)

Initialize linked list

Parametri

- **ll_p** -- pointer to *lv_ll_t* variable
- **node_size** -- the size of 1 node in bytes

void ***lv_ll_ins_head**(*lv_ll_t* *ll_p)

Add a new head to a linked list

Parametri

ll_p -- pointer to linked list

Ritorna

pointer to the new head

void *`lv_ll_ins_prev`(*lv_ll_t* *ll_p, void *n_act)

Insert a new node in front of the n_act node

Parametri

- **ll_p** -- pointer to linked list
- **n_act** -- pointer a node

Ritorna

pointer to the new node

void *`lv_ll_ins_tail`(*lv_ll_t* *ll_p)

Add a new tail to a linked list

Parametri

ll_p -- pointer to linked list

Ritorna

pointer to the new tail

void `lv_ll_remove`(*lv_ll_t* *ll_p, void *node_p)

Remove the node 'node_p' from 'll_p' linked list. It does not free the memory of node.

Parametri

- **ll_p** -- pointer to the linked list of 'node_p'
- **node_p** -- pointer to node in 'll_p' linked list

void `lv_ll_clear_custom`(*lv_ll_t* *ll_p, void (*cleanup)(void*))

void `lv_ll_clear`(*lv_ll_t* *ll_p)

Remove and free all elements from a linked list. The list remain valid but become empty.

Parametri

ll_p -- pointer to linked list

void `lv_ll_chg_list`(*lv_ll_t* *ll_ori_p, *lv_ll_t* *ll_new_p, void *node, bool head)

Move a node to a new linked list

Parametri

- **ll_ori_p** -- pointer to the original (old) linked list
- **ll_new_p** -- pointer to the new linked list
- **node** -- pointer to a node
- **head** -- true: be the head in the new list false be the tail in the new list

void *`lv_ll_get_head`(const *lv_ll_t* *ll_p)

Return with head node of the linked list

Parametri

ll_p -- pointer to linked list

Ritorna

pointer to the head of 'll_p'

void *`lv_ll_get_tail`(const *lv_ll_t* *ll_p)

Return with tail node of the linked list

Parametri

ll_p -- pointer to linked list

Ritorna

pointer to the tail of 'll_p'

void *`lv_ll_get_next`(const *lv_ll_t* *ll_p, const void *n_act)

Return with the pointer of the next node after 'n_act'

Parametri

- **ll_p** -- pointer to linked list
- **n_act** -- pointer a node

Ritorna

pointer to the next node

void *`lv_ll_get_prev`(const *lv_ll_t* *ll_p, const void *n_act)

Return with the pointer of the previous node after 'n_act'

Parametri
- **ll_p** -- pointer to linked list
- **n_act** -- pointer a node

Ritorna
pointer to the previous node

uint32_t **lv_ll_get_len**(const *lv_ll_t* *ll_p)

Return the length of the linked list.

Parametri
ll_p -- pointer to linked list

Ritorna
length of the linked list

void **lv_ll_move_before**(*lv_ll_t* *ll_p, void *n_act, void *n_after)

Move a node before another node in the same linked list

Parametri
- **ll_p** -- pointer to a linked list
- **n_act** -- pointer to node to move
- **n_after** -- pointer to a node which should be after `n_act`

bool **lv_ll_is_empty**(*lv_ll_t* *ll_p)

Check if a linked list is empty

Parametri
ll_p -- pointer to a linked list

Ritorna
true: the linked list is empty; false: not empty

struct **lv_ll_t**

#include <lv_ll.h> Description of a linked list

Public Members

uint32_t **n_size**

lv_ll_node_t ***head**

lv_ll_node_t ***tail**

14.18.22 lv_log.h

Defines

LV_LOG_LEVEL_TRACE

Log detailed information.

LV_LOG_LEVEL_INFO

Log important events.

LV_LOG_LEVEL_WARN

Log if something unwanted happened but didn't caused problem.

LV_LOG_LEVEL_ERROR

Log only critical issues, when system may fail.

LV_LOG_LEVEL_USER

Log only custom log messages added by the user.

LV_LOG_LEVEL_NONE

Do not log anything.

LV_LOG_LEVEL_NUM

Number of log levels

LV_LOG_FILE

LV_LOG_LINE

LV_LOG_TRACE(...)

LV_LOG_INFO(...)

LV_LOG_WARN(...)

LV_LOG_ERROR(...)

LV_LOG_USER(...)

LV_LOG(...)

Typedef

typedef int8_t **lv_log_level_t**

typedef void (*__lv_log_print_g_cb_t__)(*lv_log_level_t* level, const char *buf)

Log print function. Receives a string buffer to print".

Functions

void **lv_log_register_print_cb**(*lv_log_print_g_cb_t* print_cb)

Register custom print/write function to call when a log is added. It can format its "File path", "Line number" and "Description" as required and send the formatted log message to a console or serial port.

Parametri

print_cb -- a function pointer to print a log

void **lv_log**(const char *format, ...)

Print a log message via `printf` if enabled with `LV_LOG_PRINTF` in `lv_conf.h` and/or a print callback if registered with `lv_log_register_print_cb`

Parametri

- **format** -- printf-like format string
- **...** -- parameters for `format`

void **lv_log_add**(*lv_log_level_t* level, const char *file, int line, const char *func, const char *format, ...)

Add a log

Parametri

- **level** -- the level of log. (From `lv_log_level_t` enum)
- **file** -- name of the file when the log added
- **line** -- line number in the source code where the log added
- **func** -- name of the function when the log added
- **format** -- printf-like format string
- **...** -- parameters for `format`

14.18.23 lv_lru.h

Typedef

typedef void (*`lv_lru_free_cb_t`)(void *v)

typedef struct _lv_lru_item_t `lv_lru_item_t`

typedef struct *_lv_lru_t* `lv_lru_t`

Enums

enum `lv_lru_res_t`

Values:

enumerator `LV_LRU_OK`

enumerator `LV_LRU_MISSING_CACHE`

enumerator `LV_LRU_MISSING_KEY`

enumerator `LV_LRU_MISSING_VALUE`

enumerator `LV_LRU_LOCK_ERROR`

enumerator `LV_LRU_VALUE_TOO_LARGE`

Functions

lv_lru_t *`lv_lru_create`(size_t cache_size, size_t average_length, *lv_lru_free_cb_t* value_free, *lv_lru_free_cb_t* key_free)

void `lv_lru_delete`(*lv_lru_t* *cache)

lv_lru_res_t `lv_lru_set`(*lv_lru_t* *cache, const void *key, size_t key_length, void *value, size_t value_length)

lv_lru_res_t `lv_lru_get`(*lv_lru_t* *cache, const void *key, size_t key_size, void **value)

lv_lru_res_t `lv_lru_remove`(*lv_lru_t* *cache, const void *key, size_t key_size)

void `lv_lru_remove_lru_item`(*lv_lru_t* *cache)

remove the least recently used item

Todo:

we can optimise this by finding the n lru items, where n = required_space / average_length

struct `_lv_lru_t`

Public Members

lv_lru_item_t **`items`

uint64_t **access_count**

size_t **free_memory**

size_t **total_memory**

size_t **average_item_length**

size_t **hash_table_size**

uint32_t **seed**

lv_lru_free_cb_t **value_free**

lv_lru_free_cb_t **key_free**

lv_lru_item_t ***free_items**

14.18.24 lv_math.h

Defines

LV_TRIGO_SIN_MAX

LV_TRIGO_SHIFT

LV_TRIGO_SHIFT to normalize

LV_BEZIER_VAL_SHIFT

log2(LV_BEZIER_VAL_MAX): used to normalize up scaled values

LV_BEZIER_VAL_MAX

Max time in Bezier functions (not [0..1] to use integers)

LV_BEZIER_VAL_FLOAT(f)

Convert const float number cubic-bezier values to fix-point value

LV_ALIGN_UP(x, align)

Align up value x to align, align must be a power of two

LV_ROUND_UP(x, round)

Round up value x to round, round can be any integer number

LV_MIN(a, b)

LV_MIN3(a, b, c)

LV_MIN4(a, b, c, d)

LV_MAX(a, b)

LV_MAX3(a, b, c)

LV_MAX4(a, b, c, d)

LV_CLAMP(min, val, max)

LV_ABS(x)

LV_UDIV255(x)

LV_IS_SIGNED(t)

LV_UMAX_OF(t)

LV_SMAX_OF(t)

LV_MAX_OF(t)

Functions

int32_t **lv_trigo_sin**(int16_t angle)

Return with sinus of an angle

Parametri

angle --

Ritorna

sinus of 'angle'. sin(-90) = -32767, sin(90) = 32767

int32_t **lv_trigo_cos**(int16_t angle)

int32_t **lv_cubic_bezier**(int32_t x, int32_t x1, int32_t y1, int32_t x2, int32_t y2)

Calculate the y value of cubic-bezier(x1, y1, x2, y2) function as specified x.

Parametri

- **x** -- time in range of [0..LV_BEZIER_VAL_MAX]
- **x1** -- x of control point 1 in range of [0..LV_BEZIER_VAL_MAX]
- **y1** -- y of control point 1 in range of [0..LV_BEZIER_VAL_MAX]
- **x2** -- x of control point 2 in range of [0..LV_BEZIER_VAL_MAX]
- **y2** -- y of control point 2 in range of [0..LV_BEZIER_VAL_MAX]

Ritorna

the value calculated

int32_t **lv_bezier3**(int32_t t, int32_t u0, uint32_t u1, int32_t u2, int32_t u3)

Calculate a value of a Cubic Bezier function.

Parametri

- **t** -- time in range of [0..LV_BEZIER_VAL_MAX]
- **u0** -- must be 0
- **u1** -- control value 1 values in range of [0..LV_BEZIER_VAL_MAX]
- **u2** -- control value 2 in range of [0..LV_BEZIER_VAL_MAX]
- **u3** -- must be LV_BEZIER_VAL_MAX

Ritorna

the value calculated from the given parameters in range of [0..LV_BEZIER_VAL_MAX]

uint16_t **lv_atan2**(int x, int y)

Calculate the atan2 of a vector.

Parametri

- **x** --
- **y** --

Ritorna

the angle in degree calculated from the given parameters in range of [0..360]

void **lv_sqrt**(uint32_t x, *lv_sqrt_res_t* *q, uint32_t mask)

Get the square root of a number

Parametri

- **x** -- integer which square root should be calculated

- **q** -- store the result here. q->i: integer part, q->f: fractional part in 1/256 unit
- **mask** -- optional to skip some iterations if the magnitude of the root is known. Set to 0x8000 by default. If root < 16: mask = 0x80 If root < 256: mask = 0x800 Else: mask = 0x8000

int32_t **lv_sqrt32**(uint32_t x)

Alternative (fast, approximate) implementation for getting the square root of an integer.

Parametri

x -- integer which square root should be calculated

static inline int32_t **lv_sqr**(int32_t x)

Calculate the square of an integer (input range is 0..32767).

Parametri

x -- input

Ritorna

square

int64_t **lv_pow**(int64_t base, int8_t exp)

Calculate the integer exponents.

Parametri

- **base** --
- **exp** --

Ritorna

base raised to the power exponent

int32_t **lv_map**(int32_t x, int32_t min_in, int32_t max_in, int32_t min_out, int32_t max_out)

Get the mapped of a number given an input and output range

Parametri

- **x** -- integer which mapped value should be calculated
- **min_in** -- min input range
- **max_in** -- max input range
- **min_out** -- max output range
- **max_out** -- max output range

Ritorna

the mapped number

void **lv_rand_set_seed**(uint32_t seed)

Set the seed of the pseudo random number generator

Parametri

seed -- a number to initialize the random generator

uint32_t **lv_rand**(uint32_t min, uint32_t max)

Get a pseudo random number in the given range

Parametri

- **min** -- the minimum value
- **max** -- the maximum value

Ritorna

return the random number. min <= return_value <= max

struct **lv_sqrt_res_t**

Public Members

uint16_t **i**

uint16_t **f**

14.18.25 lv_matrix.h

Functions

void **lv_matrix_identity**(lv_matrix_t *matrix)

Set matrix to identity matrix

Parametri

matrix -- pointer to a matrix

void **lv_matrix_translate**(lv_matrix_t *matrix, float tx, float ty)

Translate the matrix to new position

Parametri

- **matrix** -- pointer to a matrix
- **tx** -- the amount of translate in x direction
- **tx** -- the amount of translate in y direction

void **lv_matrix_scale**(lv_matrix_t *matrix, float scale_x, float scale_y)

Change the scale factor of the matrix

Parametri

- **matrix** -- pointer to a matrix
- **scale_x** -- the scale factor for the X direction
- **scale_y** -- the scale factor for the Y direction

void **lv_matrix_rotate**(lv_matrix_t *matrix, float degree)

Rotate the matrix with origin

Parametri

- **matrix** -- pointer to a matrix
- **degree** -- angle to rotate

void **lv_matrix_skew**(lv_matrix_t *matrix, float skew_x, float skew_y)

Change the skew factor of the matrix

Parametri

- **matrix** -- pointer to a matrix
- **skew_x** -- the skew factor for x direction
- **skew_y** -- the skew factor for y direction

void **lv_matrix_multiply**(lv_matrix_t *matrix, const lv_matrix_t *mul)

Multiply two matrix and store the result to the first one

Parametri

- **matrix** -- pointer to a matrix
- **matrix2** -- pointer to another matrix

bool **lv_matrix_inverse**(lv_matrix_t *matrix, const lv_matrix_t *m)

Invert the matrix

Parametri

- **matrix** -- pointer to a matrix
- **m** -- pointer to another matrix (optional)

Ritorna

true: the matrix is invertible, false: the matrix is singular and cannot be inverted

lv_point_precise_t **lv_matrix_transform_precise_point**(const lv_matrix_t *matrix, const *lv_point_precise_t* *point)

Transform a point by a matrix

Parametri

- **matrix** -- pointer to a matrix
- **point** -- pointer to a point

Ritorna

the transformed point

lv_area_t **lv_matrix_transform_area**(const lv_matrix_t *matrix, const *lv_area_t* *area)

Transform an area by a matrix

Parametri
- **matrix** -- pointer to a matrix
- **area** -- pointer to an area

Ritorna
the transformed area

bool **lv_matrix_is_identity**(const lv_matrix_t *matrix)

Check if the matrix is identity

Parametri
matrix -- pointer to a matrix

Ritorna
true: the matrix is identity , false: the matrix is not identity

bool **lv_matrix_is_identity_or_translation**(const lv_matrix_t *matrix)

Check if the matrix is identity or translation matrix

Parametri
matrix -- pointer to a matrix

Ritorna
true: the matrix is identity or translation matrix, false: the matrix is not identity or translation matrix

struct **_lv_matrix_t**

Public Members

float **m**[3][3]

14.18.26 lv_palette.h

Enums

enum **lv_palette_t**

Values:

enumerator **LV_PALETTE_RED**

enumerator **LV_PALETTE_PINK**

enumerator **LV_PALETTE_PURPLE**

enumerator **LV_PALETTE_DEEP_PURPLE**

enumerator **LV_PALETTE_INDIGO**

enumerator **LV_PALETTE_BLUE**

enumerator **LV_PALETTE_LIGHT_BLUE**

enumerator **LV_PALETTE_CYAN**

enumerator **LV_PALETTE_TEAL**

enumerator **LV_PALETTE_GREEN**

enumerator **LV_PALETTE_LIGHT_GREEN**

enumerator **LV_PALETTE_LIME**

enumerator **LV_PALETTE_YELLOW**

enumerator **LV_PALETTE_AMBER**

enumerator **LV_PALETTE_ORANGE**

enumerator **LV_PALETTE_DEEP_ORANGE**

enumerator **LV_PALETTE_BROWN**

enumerator **LV_PALETTE_BLUE_GREY**

enumerator **LV_PALETTE_GREY**

enumerator **LV_PALETTE_LAST**

enumerator **LV_PALETTE_NONE**

Functions

lv_color_t **lv_palette_main**(*lv_palette_t* p)

lv_color_t **lv_palette_lighten**(*lv_palette_t* p, uint8_t lvl)

lv_color_t **lv_palette_darken**(*lv_palette_t* p, uint8_t lvl)

14.18.27 lv_profiler.h

Defines

LV_PROFILER_BEGIN

LV_PROFILER_END

LV_PROFILER_BEGIN_TAG(tag)

LV_PROFILER_END_TAG(tag)

LV_PROFILER_LAYOUT_BEGIN

LV_PROFILER_LAYOUT_END

LV_PROFILER_LAYOUT_BEGIN_TAG(tag)

LV_PROFILER_LAYOUT_END_TAG(tag)

LV_PROFILER_STYLE_BEGIN

LV_PROFILER_STYLE_END

LV_PROFILER_STYLE_BEGIN_TAG(tag)

LV_PROFILER_STYLE_END_TAG(tag)

LV_PROFILER_DRAW_BEGIN

LV_PROFILER_DRAW_END

LV_PROFILER_DRAW_BEGIN_TAG(tag)

LV_PROFILER_DRAW_END_TAG(tag)

LV_PROFILER_DECODER_BEGIN

LV_PROFILER_DECODER_END

LV_PROFILER_DECODER_BEGIN_TAG(tag)

LV_PROFILER_DECODER_END_TAG(tag)

LV_PROFILER_REFR_BEGIN

LV_PROFILER_REFR_END

LV_PROFILER_REFR_BEGIN_TAG(tag)

LV_PROFILER_REFR_END_TAG(tag)

LV_PROFILER_INDEV_BEGIN

LV_PROFILER_INDEV_END

LV_PROFILER_INDEV_BEGIN_TAG(tag)

LV_PROFILER_INDEV_END_TAG(tag)

LV_PROFILER_FONT_BEGIN

LV_PROFILER_FONT_END

LV_PROFILER_FONT_BEGIN_TAG(tag)

LV_PROFILER_FONT_END_TAG(tag)

LV_PROFILER_CACHE_BEGIN

LV_PROFILER_CACHE_END

LV_PROFILER_CACHE_BEGIN_TAG(tag)

LV_PROFILER_CACHE_END_TAG(tag)

LV_PROFILER_FS_BEGIN

LV_PROFILER_FS_END

LV_PROFILER_FS_BEGIN_TAG(tag)

LV_PROFILER_FS_END_TAG(tag)

LV_PROFILER_TIMER_BEGIN

LV_PROFILER_TIMER_END

LV_PROFILER_TIMER_BEGIN_TAG(tag)

LV_PROFILER_TIMER_END_TAG(tag)

LV_PROFILER_EVENT_BEGIN

LV_PROFILER_EVENT_END

LV_PROFILER_EVENT_BEGIN_TAG(tag)

LV_PROFILER_EVENT_END_TAG(tag)

14.18.28 lv_profiler_builtin.h

14.18.29 lv_profiler_builtin_private.h

14.18.30 lv_rb.h

Typedef

typedef int8_t **lv_rb_compare_res_t**

typedef *lv_rb_compare_res_t* (*__lv_rb_compare_t__)(const void *a, const void *b)

Enums

enum **lv_rb_color_t**

Values:

enumerator **LV_RB_COLOR_RED**

enumerator **LV_RB_COLOR_BLACK**

Functions

bool **lv_rb_init**(lv_rb_t *tree, *lv_rb_compare_t* compare, size_t node_size)

lv_rb_node_t ***lv_rb_insert**(lv_rb_t *tree, void *key)

lv_rb_node_t ***lv_rb_find**(lv_rb_t *tree, const void *key)

void ***lv_rb_remove_node**(lv_rb_t *tree, lv_rb_node_t *node)

void ***lv_rb_remove**(lv_rb_t *tree, const void *key)

bool **lv_rb_drop_node**(lv_rb_t *tree, lv_rb_node_t *node)

bool **lv_rb_drop**(lv_rb_t *tree, const void *key)

lv_rb_node_t ***lv_rb_minimum**(lv_rb_t *node)

lv_rb_node_t ***lv_rb_maximum**(lv_rb_t *node)

lv_rb_node_t ***lv_rb_minimum_from**(lv_rb_node_t *node)

lv_rb_node_t ***lv_rb_maximum_from**(lv_rb_node_t *node)

void **lv_rb_destroy**(lv_rb_t *tree)

14.18.31 lv_rb_private.h

struct **_lv_rb_node_t**

Public Members

struct *_lv_rb_node_t* ***parent**

struct *_lv_rb_node_t* ***left**

struct *_lv_rb_node_t* ***right**

lv_rb_color_t **color**

void ***data**

struct **_lv_rb_t**

Public Members

lv_rb_node_t ***root**

lv_rb_compare_t **compare**

size_t **size**

14.18.32 lv_style.h

Defines

LV_STYLE_SENTINEL_VALUE

LV_STYLE_PROP_FLAG_NONE

No special behavior

LV_STYLE_PROP_FLAG_INHERITABLE

Inherited

LV_STYLE_PROP_FLAG_EXT_DRAW_UPDATE

Requires ext. draw size update when changed

LV_STYLE_PROP_FLAG_LAYOUT_UPDATE

Requires layout update when changed

LV_STYLE_PROP_FLAG_PARENT_LAYOUT_UPDATE

Requires layout update on parent when changed

LV_STYLE_PROP_FLAG_LAYER_UPDATE

Affects layer handling

LV_STYLE_PROP_FLAG_TRANSFORM

Affects the object's transformation

LV_STYLE_PROP_FLAG_ALL

Indicating all flags

LV_SCALE_NONE

Value for not zooming the image

LV_STYLE_CONST_INIT(var_name, prop_array)

LV_STYLE_CONST_PROPS_END

LV_GRAD_LEFT

LV_GRAD_RIGHT

LV_GRAD_TOP

LV_GRAD_BOTTOM

LV_GRAD_CENTER

LV_ASSERT_STYLE(style_p)

Enums

enum **lv_blend_mode_t**

Possible options for blending opaque drawings

Values:

enumerator **LV_BLEND_MODE_NORMAL**

Simply mix according to the opacity value

enumerator **LV_BLEND_MODE_ADDITIVE**

Add the respective color channels

enumerator **LV_BLEND_MODE_SUBTRACTIVE**

Subtract the foreground from the background

enumerator **LV_BLEND_MODE_MULTIPLY**

Multiply the foreground and background

enumerator **LV_BLEND_MODE_DIFFERENCE**

Absolute difference between foreground and background

enum **lv_text_decor_t**

Some options to apply decorations on texts. 'OR'ed values can be used.

Values:

enumerator **LV_TEXT_DECOR_NONE**

enumerator **LV_TEXT_DECOR_UNDERLINE**

enumerator **LV_TEXT_DECOR_STRIKETHROUGH**

enum **lv_border_side_t**

Selects on which sides border should be drawn 'OR'ed values can be used.

Values:

enumerator **LV_BORDER_SIDE_NONE**

enumerator **LV_BORDER_SIDE_BOTTOM**

enumerator **LV_BORDER_SIDE_TOP**

enumerator **LV_BORDER_SIDE_LEFT**

enumerator **LV_BORDER_SIDE_RIGHT**

enumerator **LV_BORDER_SIDE_FULL**

enumerator **LV_BORDER_SIDE_INTERNAL**

FOR matrix-like objects (e.g. Button matrix)

enum **_lv_style_id_t**

Enumeration of all built in style properties

Props are split into groups of 16. When adding a new prop to a group, ensure it does not overflow into the next one.

Values:

enumerator **LV_STYLE_PROP_INV**

enumerator **LV_STYLE_WIDTH**

enumerator **LV_STYLE_HEIGHT**

enumerator **LV_STYLE_LENGTH**

enumerator **LV_STYLE_MIN_WIDTH**

enumerator **LV_STYLE_MAX_WIDTH**

enumerator **LV_STYLE_MIN_HEIGHT**

enumerator **LV_STYLE_MAX_HEIGHT**

enumerator **LV_STYLE_X**

enumerator **LV_STYLE_Y**

enumerator **LV_STYLE_ALIGN**

enumerator **LV_STYLE_RADIUS**

enumerator **LV_STYLE_RADIAL_OFFSET**

enumerator **LV_STYLE_PAD_RADIAL**

enumerator **LV_STYLE_PAD_TOP**

enumerator **LV_STYLE_PAD_BOTTOM**

enumerator **LV_STYLE_PAD_LEFT**

enumerator **LV_STYLE_PAD_RIGHT**

enumerator **LV_STYLE_PAD_ROW**

enumerator **LV_STYLE_PAD_COLUMN**

enumerator **LV_STYLE_LAYOUT**

enumerator **LV_STYLE_MARGIN_TOP**

enumerator **LV_STYLE_MARGIN_BOTTOM**

enumerator **LV_STYLE_MARGIN_LEFT**

enumerator **LV_STYLE_MARGIN_RIGHT**

enumerator **LV_STYLE_BG_COLOR**

enumerator **LV_STYLE_BG_OPA**

enumerator **LV_STYLE_BG_GRAD_DIR**

enumerator **LV_STYLE_BG_MAIN_STOP**

enumerator **LV_STYLE_BG_GRAD_STOP**

enumerator **LV_STYLE_BG_GRAD_COLOR**

enumerator **LV_STYLE_BG_MAIN_OPA**

enumerator **LV_STYLE_BG_GRAD_OPA**

enumerator **LV_STYLE_BG_GRAD**

enumerator **LV_STYLE_BASE_DIR**

enumerator **LV_STYLE_BG_IMAGE_SRC**

enumerator **LV_STYLE_BG_IMAGE_OPA**

enumerator **LV_STYLE_BG_IMAGE_RECOLOR**

enumerator **LV_STYLE_BG_IMAGE_RECOLOR_OPA**

enumerator **LV_STYLE_BG_IMAGE_TILED**

enumerator **LV_STYLE_CLIP_CORNER**

enumerator **LV_STYLE_BORDER_WIDTH**

enumerator **LV_STYLE_BORDER_COLOR**

enumerator **LV_STYLE_BORDER_OPA**

enumerator **LV_STYLE_BORDER_SIDE**

enumerator **LV_STYLE_BORDER_POST**

enumerator **LV_STYLE_OUTLINE_WIDTH**

enumerator **LV_STYLE_OUTLINE_COLOR**

enumerator **LV_STYLE_OUTLINE_OPA**

enumerator **LV_STYLE_OUTLINE_PAD**

enumerator **LV_STYLE_SHADOW_WIDTH**

enumerator **LV_STYLE_SHADOW_COLOR**

enumerator **LV_STYLE_SHADOW_OPA**

enumerator **LV_STYLE_SHADOW_OFFSET_X**

enumerator **LV_STYLE_SHADOW_OFFSET_Y**

enumerator **LV_STYLE_SHADOW_SPREAD**

enumerator **LV_STYLE_IMAGE_OPA**

enumerator **LV_STYLE_IMAGE_RECOLOR**

enumerator **LV_STYLE_IMAGE_RECOLOR_OPA**

enumerator **LV_STYLE_LINE_WIDTH**

enumerator **LV_STYLE_LINE_DASH_WIDTH**

enumerator **LV_STYLE_LINE_DASH_GAP**

enumerator **LV_STYLE_LINE_ROUNDED**

enumerator **LV_STYLE_LINE_COLOR**

enumerator **LV_STYLE_LINE_OPA**

enumerator **LV_STYLE_ARC_WIDTH**

enumerator **LV_STYLE_ARC_ROUNDED**

enumerator **LV_STYLE_ARC_COLOR**

enumerator **LV_STYLE_ARC_OPA**

enumerator **LV_STYLE_ARC_IMAGE_SRC**

enumerator **LV_STYLE_TEXT_COLOR**

enumerator **LV_STYLE_TEXT_OPA**

enumerator **LV_STYLE_TEXT_FONT**

enumerator **LV_STYLE_TEXT_LETTER_SPACE**

enumerator **LV_STYLE_TEXT_LINE_SPACE**

enumerator **LV_STYLE_TEXT_DECOR**

enumerator **LV_STYLE_TEXT_ALIGN**

enumerator **LV_STYLE_TEXT_OUTLINE_STROKE_WIDTH**

enumerator **LV_STYLE_TEXT_OUTLINE_STROKE_OPA**

enumerator **LV_STYLE_TEXT_OUTLINE_STROKE_COLOR**

enumerator **LV_STYLE_OPA**

enumerator **LV_STYLE_OPA_LAYERED**

enumerator **LV_STYLE_COLOR_FILTER_DSC**

enumerator **LV_STYLE_COLOR_FILTER_OPA**

enumerator **LV_STYLE_ANIM**

enumerator **LV_STYLE_ANIM_DURATION**

enumerator **LV_STYLE_TRANSITION**

enumerator **LV_STYLE_BLEND_MODE**

enumerator **LV_STYLE_TRANSFORM_WIDTH**

enumerator **LV_STYLE_TRANSFORM_HEIGHT**

enumerator **LV_STYLE_TRANSLATE_X**

enumerator **LV_STYLE_TRANSLATE_Y**

enumerator **LV_STYLE_TRANSFORM_SCALE_X**

enumerator **LV_STYLE_TRANSFORM_SCALE_Y**

enumerator **LV_STYLE_TRANSFORM_ROTATION**

enumerator **LV_STYLE_TRANSFORM_PIVOT_X**

enumerator **LV_STYLE_TRANSFORM_PIVOT_Y**

enumerator **LV_STYLE_TRANSFORM_SKEW_X**

enumerator **LV_STYLE_TRANSFORM_SKEW_Y**

enumerator **LV_STYLE_BITMAP_MASK_SRC**

enumerator **LV_STYLE_ROTARY_SENSITIVITY**

enumerator **LV_STYLE_TRANSLATE_RADIAL**

enumerator **LV_STYLE_RECOLOR**

enumerator **LV_STYLE_RECOLOR_OPA**

enumerator **LV_STYLE_FLEX_FLOW**

enumerator **LV_STYLE_FLEX_MAIN_PLACE**

enumerator **LV_STYLE_FLEX_CROSS_PLACE**

enumerator **LV_STYLE_FLEX_TRACK_PLACE**

enumerator **LV_STYLE_FLEX_GROW**

enumerator **LV_STYLE_GRID_COLUMN_ALIGN**

enumerator **LV_STYLE_GRID_ROW_ALIGN**

enumerator **LV_STYLE_GRID_ROW_DSC_ARRAY**

enumerator **LV_STYLE_GRID_COLUMN_DSC_ARRAY**

enumerator **LV_STYLE_GRID_CELL_COLUMN_POS**

enumerator **LV_STYLE_GRID_CELL_COLUMN_SPAN**

enumerator **LV_STYLE_GRID_CELL_X_ALIGN**

enumerator **LV_STYLE_GRID_CELL_ROW_POS**

enumerator **LV_STYLE_GRID_CELL_ROW_SPAN**

enumerator **LV_STYLE_GRID_CELL_Y_ALIGN**

enumerator **LV_STYLE_LAST_BUILT_IN_PROP**

enumerator **LV_STYLE_NUM_BUILT_IN_PROPS**

enumerator **LV_STYLE_PROP_ANY**

enumerator **LV_STYLE_PROP_CONST**

enum **lv_style_res_t**

Values:

enumerator **LV_STYLE_RES_NOT_FOUND**

enumerator **LV_STYLE_RES_FOUND**

Functions

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

Initialize a style

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

Parametri

style -- pointer to a style to initialize

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

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

Parametri

style -- pointer to a style

void **lv_style_copy**(*lv_style_t* *dst, const *lv_style_t* *src)

Copy all properties of a style to an other. It has the same affect callying the same `lv_set_style_...` functions on both styles. It means new memory will be allocated to store the properties in the destination style. After the copy the destination style is fully independent of the source and source can removed without affecting the destination style.

Parametri

- **dst** -- the destination to copy into (can not the a constant style)
- **src** -- the source style to copy from.

static inline bool **lv_style_is_const**(const *lv_style_t* *style)

Check if a style is constant

Parametri

style -- pointer to a style

Ritorna

true: the style is constant

lv_style_prop_t **lv_style_register_prop**(uint8_t flag)

Register a new style property for custom usage

Example:

```
lv_style_prop_t MY_PROP;
static inline void lv_style_set_my_prop(lv_style_t * style, lv_color_t␣
↪value) {
lv_style_value_t v = {.color = value}; lv_style_set_prop(style, MY_
↪PROP, v); }

...
MY_PROP = lv_style_register_prop();
...
lv_style_set_my_prop(&style1, lv_palette_main(LV_PALETTE_RED));
```

Ritorna

a new property ID, or LV_STYLE_PROP_INV if there are no more available.

lv_style_prop_t **lv_style_get_num_custom_props**(void)

Get the number of custom properties that have been registered thus far.

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

Remove a property from a style

Parametri

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

Ritorna

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

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

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

Parametri

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

lv_style_res_t **lv_style_get_prop**(const *lv_style_t* *style, lv_style_prop_t prop, *lv_style_value_t* *value)

Get the value of a property

> **Nota**
>
> For performance reasons there are no sanity check on `style`

Parametri

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

Ritorna

LV_RESULT_INVALID: the property wasn't found in the style (`value` is unchanged)
LV_RESULT_OK: the property was fond, and `value` is set accordingly

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

Initialize a transition descriptor.

Example:

```
const static lv_style_prop_t trans_props[] = { LV_STYLE_BG_OPA, LV_
↪STYLE_BG_COLOR, 0 };
static lv_style_transition_dsc_t trans1;
lv_style_transition_dsc_init(&trans1, trans_props, NULL, 300, 0, NULL);
```

Parametri

- **tr** -- pointer to a transition descriptor to initialize
- **props** -- an array with the properties to transition. The last element must be zero.
- **path_cb** -- an animation path (ease) callback. If `NULL` liner path will be used.
- **time** -- duration of the transition in [ms]
- **delay** -- delay before the transition in [ms]
- **user_data** -- any custom data that will be saved in the transition animation and will be available when `path_cb` is called

lv_style_value_t **lv_style_prop_get_default**(lv_style_prop_t prop)

Get the default value of a property

Parametri

prop -- the ID of a property

Ritorna

the default value

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

Get the value of a property

> **Nota**
>
> For performance reasons there are no sanity check on `style`

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

Parametri

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

Ritorna

LV_RESULT_INVALID: the property wasn't found in the style (`value` is unchanged)
LV_RESULT_OK: the property was fond, and `value` is set accordingly

bool **lv_style_is_empty**(const *lv_style_t* *style)

Checks if a style is empty (has no properties)

Parametri

style -- pointer to a style

Ritorna

true if the style is empty

static inline uint32_t **lv_style_get_prop_group**(lv_style_prop_t prop)

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

Parametri

prop -- a style property

Ritorna

the group [0..30] 30 means all the custom properties with index > 120

uint8_t **lv_style_prop_lookup_flags**(lv_style_prop_t prop)

Get the flags of a built-in or custom property.

Parametri

prop -- a style property

Ritorna

the flags of the property

static inline void **lv_style_set_size**(*lv_style_t* *style, int32_t width, int32_t height)

static inline void **lv_style_set_pad_all**(*lv_style_t* *style, int32_t value)

static inline void **lv_style_set_pad_hor**(*lv_style_t* *style, int32_t value)

static inline void **lv_style_set_pad_ver**(*lv_style_t* *style, int32_t value)

static inline void **lv_style_set_pad_gap**(*lv_style_t* *style, int32_t value)

static inline void **lv_style_set_margin_hor**(*lv_style_t* *style, int32_t value)

static inline void **lv_style_set_margin_ver**(*lv_style_t* *style, int32_t value)

static inline void **lv_style_set_margin_all**(*lv_style_t* *style, int32_t value)

static inline void **lv_style_set_transform_scale**(*lv_style_t* *style, int32_t value)

static inline bool **lv_style_prop_has_flag**(lv_style_prop_t prop, uint8_t flag)

Check if the style property has a specified behavioral flag.

Do not pass multiple flags to this function as backwards-compatibility is not guaranteed for that.

Parametri

- **prop** -- Property ID
- **flag** -- Flag

Ritorna

true if the flag is set for this property

Variables

const lv_style_prop_t **lv_style_const_prop_id_inv**

union **lv_style_value_t**

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

Public Members

int32_t **num**

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

const void ***ptr**

Constant pointers (font, cone text, etc)

lv_color_t **color**

Colors

struct **lv_style_transition_dsc_t**

#include <lv_style.h> Descriptor for style transitions

Public Members

const lv_style_prop_t ***props**

An array with the properties to animate.

void ***user_data**

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

lv_anim_path_cb_t **path_xcb**

A path for the animation.

uint32_t **time**

Duration of the transition in [ms]

uint32_t **delay**

Delay before the transition in [ms]

struct **lv_style_const_prop_t**

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

Public Members

lv_style_prop_t **prop**

lv_style_value_t **value**

struct **lv_style_t**

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

Public Members

uint32_t **sentinel**

void ***values_and_props**

uint32_t **has_group**

uint8_t **prop_cnt**

255 means it's a constant style

14.18.33 lv_style_gen.h

Defines

LV_STYLE_CONST_WIDTH(val)

LV_STYLE_CONST_MIN_WIDTH(val)

LV_STYLE_CONST_MAX_WIDTH(val)

LV_STYLE_CONST_HEIGHT(val)

LV_STYLE_CONST_MIN_HEIGHT(val)

LV_STYLE_CONST_MAX_HEIGHT(val)

LV_STYLE_CONST_LENGTH(val)

LV_STYLE_CONST_X(val)

LV_STYLE_CONST_Y(val)

LV_STYLE_CONST_ALIGN(val)

LV_STYLE_CONST_TRANSFORM_WIDTH(val)

LV_STYLE_CONST_TRANSFORM_HEIGHT(val)

LV_STYLE_CONST_TRANSLATE_X(val)

LV_STYLE_CONST_TRANSLATE_Y(val)

LV_STYLE_CONST_TRANSLATE_RADIAL(val)

LV_STYLE_CONST_TRANSFORM_SCALE_X(val)

LV_STYLE_CONST_TRANSFORM_SCALE_Y(val)

LV_STYLE_CONST_TRANSFORM_ROTATION(val)

LV_STYLE_CONST_TRANSFORM_PIVOT_X(val)

LV_STYLE_CONST_TRANSFORM_PIVOT_Y(val)

LV_STYLE_CONST_TRANSFORM_SKEW_X(val)

LV_STYLE_CONST_TRANSFORM_SKEW_Y(val)

LV_STYLE_CONST_PAD_TOP(val)

LV_STYLE_CONST_PAD_BOTTOM(val)

LV_STYLE_CONST_PAD_LEFT(val)

LV_STYLE_CONST_PAD_RIGHT(val)

LV_STYLE_CONST_PAD_ROW(val)

LV_STYLE_CONST_PAD_COLUMN(val)

LV_STYLE_CONST_PAD_RADIAL(val)

LV_STYLE_CONST_MARGIN_TOP(val)

LV_STYLE_CONST_MARGIN_BOTTOM(val)

LV_STYLE_CONST_MARGIN_LEFT(val)

LV_STYLE_CONST_MARGIN_RIGHT(val)

LV_STYLE_CONST_BG_COLOR(val)

LV_STYLE_CONST_BG_OPA(val)

LV_STYLE_CONST_BG_GRAD_COLOR(val)

LV_STYLE_CONST_BG_GRAD_DIR(val)

LV_STYLE_CONST_BG_MAIN_STOP(val)

LV_STYLE_CONST_BG_GRAD_STOP(val)

LV_STYLE_CONST_BG_MAIN_OPA(val)

LV_STYLE_CONST_BG_GRAD_OPA(val)

LV_STYLE_CONST_BG_GRAD(val)

LV_STYLE_CONST_BG_IMAGE_SRC(val)

LV_STYLE_CONST_BG_IMAGE_OPA(val)

LV_STYLE_CONST_BG_IMAGE_RECOLOR(val)

LV_STYLE_CONST_BG_IMAGE_RECOLOR_OPA(val)

LV_STYLE_CONST_BG_IMAGE_TILED(val)

LV_STYLE_CONST_BORDER_COLOR(val)

LV_STYLE_CONST_BORDER_OPA(val)

LV_STYLE_CONST_BORDER_WIDTH(val)

LV_STYLE_CONST_BORDER_SIDE(val)

LV_STYLE_CONST_BORDER_POST(val)

LV_STYLE_CONST_OUTLINE_WIDTH(val)

LV_STYLE_CONST_OUTLINE_COLOR(val)

LV_STYLE_CONST_OUTLINE_OPA(val)

LV_STYLE_CONST_OUTLINE_PAD(val)

LV_STYLE_CONST_SHADOW_WIDTH(val)

LV_STYLE_CONST_SHADOW_OFFSET_X(val)

LV_STYLE_CONST_SHADOW_OFFSET_Y(val)

LV_STYLE_CONST_SHADOW_SPREAD(val)

LV_STYLE_CONST_SHADOW_COLOR(val)

LV_STYLE_CONST_SHADOW_OPA(val)

LV_STYLE_CONST_IMAGE_OPA(val)

LV_STYLE_CONST_IMAGE_RECOLOR(val)

LV_STYLE_CONST_IMAGE_RECOLOR_OPA(val)

LV_STYLE_CONST_LINE_WIDTH(val)

LV_STYLE_CONST_LINE_DASH_WIDTH(val)

LV_STYLE_CONST_LINE_DASH_GAP(val)

LV_STYLE_CONST_LINE_ROUNDED(val)

LV_STYLE_CONST_LINE_COLOR(val)

LV_STYLE_CONST_LINE_OPA(val)

LV_STYLE_CONST_ARC_WIDTH(val)

LV_STYLE_CONST_ARC_ROUNDED(val)

LV_STYLE_CONST_ARC_COLOR(val)

LV_STYLE_CONST_ARC_OPA(val)

LV_STYLE_CONST_ARC_IMAGE_SRC(val)

LV_STYLE_CONST_TEXT_COLOR(val)

LV_STYLE_CONST_TEXT_OPA(val)

LV_STYLE_CONST_TEXT_FONT(val)

LV_STYLE_CONST_TEXT_LETTER_SPACE(val)

LV_STYLE_CONST_TEXT_LINE_SPACE(val)

LV_STYLE_CONST_TEXT_DECOR(val)

LV_STYLE_CONST_TEXT_ALIGN(val)

LV_STYLE_CONST_TEXT_OUTLINE_STROKE_COLOR(val)

LV_STYLE_CONST_TEXT_OUTLINE_STROKE_WIDTH(val)

LV_STYLE_CONST_TEXT_OUTLINE_STROKE_OPA(val)

LV_STYLE_CONST_RADIUS(val)

LV_STYLE_CONST_RADIAL_OFFSET(val)

LV_STYLE_CONST_CLIP_CORNER(val)

LV_STYLE_CONST_OPA(val)

LV_STYLE_CONST_OPA_LAYERED(val)

LV_STYLE_CONST_COLOR_FILTER_DSC(val)

LV_STYLE_CONST_COLOR_FILTER_OPA(val)

LV_STYLE_CONST_RECOLOR(val)

LV_STYLE_CONST_RECOLOR_OPA(val)

LV_STYLE_CONST_ANIM(val)

LV_STYLE_CONST_ANIM_DURATION(val)

LV_STYLE_CONST_TRANSITION(val)

LV_STYLE_CONST_BLEND_MODE(val)

LV_STYLE_CONST_LAYOUT(val)

LV_STYLE_CONST_BASE_DIR(val)

LV_STYLE_CONST_BITMAP_MASK_SRC(val)

LV_STYLE_CONST_ROTARY_SENSITIVITY(val)

Functions

void **lv_style_set_width**(*lv_style_t* *style, int32_t value)

void **lv_style_set_min_width**(*lv_style_t* *style, int32_t value)

void **lv_style_set_max_width**(*lv_style_t* *style, int32_t value)

void **lv_style_set_height**(*lv_style_t* *style, int32_t value)

void **lv_style_set_min_height**(*lv_style_t* *style, int32_t value)

void **lv_style_set_max_height**(*lv_style_t* *style, int32_t value)

void **lv_style_set_length**(*lv_style_t* *style, int32_t value)

void **lv_style_set_x**(*lv_style_t* *style, int32_t value)

void **lv_style_set_y**(*lv_style_t* *style, int32_t value)

void **lv_style_set_align**(*lv_style_t* *style, *lv_align_t* value)

void **lv_style_set_transform_width**(*lv_style_t* *style, int32_t value)

void **lv_style_set_transform_height**(*lv_style_t* *style, int32_t value)

void **lv_style_set_translate_x**(*lv_style_t* *style, int32_t value)

void **lv_style_set_translate_y**(*lv_style_t* *style, int32_t value)

void **lv_style_set_translate_radial**(*lv_style_t* *style, int32_t value)

void **lv_style_set_transform_scale_x**(*lv_style_t* *style, int32_t value)

void **lv_style_set_transform_scale_y**(*lv_style_t* *style, int32_t value)

void **lv_style_set_transform_rotation**(*lv_style_t* *style, int32_t value)

void **lv_style_set_transform_pivot_x**(*lv_style_t* *style, int32_t value)

void **lv_style_set_transform_pivot_y**(*lv_style_t* *style, int32_t value)

void **lv_style_set_transform_skew_x**(*lv_style_t* *style, int32_t value)

void **lv_style_set_transform_skew_y**(*lv_style_t* *style, int32_t value)

void **lv_style_set_pad_top**(*lv_style_t* *style, int32_t value)

void **lv_style_set_pad_bottom**(*lv_style_t* *style, int32_t value)

void **lv_style_set_pad_left**(*lv_style_t* *style, int32_t value)

void **lv_style_set_pad_right**(*lv_style_t* *style, int32_t value)

void **lv_style_set_pad_row**(*lv_style_t* *style, int32_t value)

void **lv_style_set_pad_column**(*lv_style_t* *style, int32_t value)

void **lv_style_set_pad_radial**(*lv_style_t* *style, int32_t value)

void **lv_style_set_margin_top**(*lv_style_t* *style, int32_t value)

void **lv_style_set_margin_bottom**(*lv_style_t* *style, int32_t value)

void **lv_style_set_margin_left**(*lv_style_t* *style, int32_t value)

void **lv_style_set_margin_right**(*lv_style_t* *style, int32_t value)

void **lv_style_set_bg_color**(*lv_style_t* *style, *lv_color_t* value)

void **lv_style_set_bg_opa**(*lv_style_t* *style, lv_opa_t value)

void **lv_style_set_bg_grad_color**(*lv_style_t* *style, *lv_color_t* value)

void **lv_style_set_bg_grad_dir**(*lv_style_t* *style, *lv_grad_dir_t* value)

void **lv_style_set_bg_main_stop**(*lv_style_t* *style, int32_t value)

void **lv_style_set_bg_grad_stop**(*lv_style_t* *style, int32_t value)

void **lv_style_set_bg_main_opa**(*lv_style_t* *style, lv_opa_t value)

void **lv_style_set_bg_grad_opa**(*lv_style_t* *style, lv_opa_t value)

void **lv_style_set_bg_grad**(*lv_style_t* *style, const *lv_grad_dsc_t* *value)

void **lv_style_set_bg_image_src**(*lv_style_t* *style, const void *value)

void **lv_style_set_bg_image_opa**(*lv_style_t* *style, lv_opa_t value)

void **lv_style_set_bg_image_recolor**(*lv_style_t* *style, *lv_color_t* value)

void **lv_style_set_bg_image_recolor_opa**(*lv_style_t* *style, lv_opa_t value)

void **lv_style_set_bg_image_tiled**(*lv_style_t* *style, bool value)

void **lv_style_set_border_color**(*lv_style_t* *style, *lv_color_t* value)

void **lv_style_set_border_opa**(*lv_style_t* *style, lv_opa_t value)

void **lv_style_set_border_width**(*lv_style_t* *style, int32_t value)

void **lv_style_set_border_side**(*lv_style_t* *style, *lv_border_side_t* value)

void **lv_style_set_border_post**(*lv_style_t* *style, bool value)

void **lv_style_set_outline_width**(*lv_style_t* *style, int32_t value)

void **lv_style_set_outline_color**(*lv_style_t* *style, *lv_color_t* value)

void **lv_style_set_outline_opa**(*lv_style_t* *style, lv_opa_t value)

void **lv_style_set_outline_pad**(*lv_style_t* *style, int32_t value)

void **lv_style_set_shadow_width**(*lv_style_t* *style, int32_t value)

void **lv_style_set_shadow_offset_x**(*lv_style_t* *style, int32_t value)

void **lv_style_set_shadow_offset_y**(*lv_style_t* *style, int32_t value)

void **lv_style_set_shadow_spread**(*lv_style_t* *style, int32_t value)

void **lv_style_set_shadow_color**(*lv_style_t* *style, *lv_color_t* value)

void **lv_style_set_shadow_opa**(*lv_style_t* *style, lv_opa_t value)

void **lv_style_set_image_opa**(*lv_style_t* *style, lv_opa_t value)

void **lv_style_set_image_recolor**(*lv_style_t* *style, *lv_color_t* value)

void **lv_style_set_image_recolor_opa**(*lv_style_t* *style, lv_opa_t value)

void **lv_style_set_line_width**(*lv_style_t* *style, int32_t value)

void **lv_style_set_line_dash_width**(*lv_style_t* *style, int32_t value)

void **lv_style_set_line_dash_gap**(*lv_style_t* *style, int32_t value)

void **lv_style_set_line_rounded**(*lv_style_t* *style, bool value)

void **lv_style_set_line_color**(*lv_style_t* *style, *lv_color_t* value)

void **lv_style_set_line_opa**(*lv_style_t* *style, lv_opa_t value)

void **lv_style_set_arc_width**(*lv_style_t* *style, int32_t value)

void **lv_style_set_arc_rounded**(*lv_style_t* *style, bool value)

void **lv_style_set_arc_color**(*lv_style_t* *style, *lv_color_t* value)

void **lv_style_set_arc_opa**(*lv_style_t* *style, lv_opa_t value)

void **lv_style_set_arc_image_src**(*lv_style_t* *style, const void *value)

void **lv_style_set_text_color**(*lv_style_t* *style, *lv_color_t* value)

void **lv_style_set_text_opa**(*lv_style_t* *style, lv_opa_t value)

void **lv_style_set_text_font**(*lv_style_t* *style, const lv_font_t *value)

void **lv_style_set_text_letter_space**(*lv_style_t* *style, int32_t value)

void **lv_style_set_text_line_space**(*lv_style_t* *style, int32_t value)

void **lv_style_set_text_decor**(*lv_style_t* *style, *lv_text_decor_t* value)

void **lv_style_set_text_align**(*lv_style_t* *style, *lv_text_align_t* value)

void **lv_style_set_text_outline_stroke_color**(*lv_style_t* *style, *lv_color_t* value)

void **lv_style_set_text_outline_stroke_width**(*lv_style_t* *style, int32_t value)

void **lv_style_set_text_outline_stroke_opa**(*lv_style_t* *style, lv_opa_t value)

void **lv_style_set_radius**(*lv_style_t* *style, int32_t value)

void **lv_style_set_radial_offset**(*lv_style_t* *style, int32_t value)

void **lv_style_set_clip_corner**(*lv_style_t* *style, bool value)

void **lv_style_set_opa**(*lv_style_t* *style, lv_opa_t value)

void **lv_style_set_opa_layered**(*lv_style_t* *style, lv_opa_t value)

void **lv_style_set_color_filter_dsc**(*lv_style_t* *style, const lv_color_filter_dsc_t *value)

void **lv_style_set_color_filter_opa**(*lv_style_t* *style, lv_opa_t value)

void **lv_style_set_recolor**(*lv_style_t* *style, *lv_color_t* value)

void **lv_style_set_recolor_opa**(*lv_style_t* *style, lv_opa_t value)

void **lv_style_set_anim**(*lv_style_t* *style, const lv_anim_t *value)

void **lv_style_set_anim_duration**(*lv_style_t* *style, uint32_t value)

void **lv_style_set_transition**(*lv_style_t* *style, const *lv_style_transition_dsc_t* *value)

void **lv_style_set_blend_mode**(*lv_style_t* *style, *lv_blend_mode_t* value)

void **lv_style_set_layout**(*lv_style_t* *style, uint16_t value)

void **lv_style_set_base_dir**(*lv_style_t* *style, *lv_base_dir_t* value)

void **lv_style_set_bitmap_mask_src**(*lv_style_t* *style, const void *value)

void **lv_style_set_rotary_sensitivity**(*lv_style_t* *style, uint32_t value)

14.18.34 lv_style_private.h

14.18.35 lv_templ.h

14.18.36 lv_text.h

Defines

LV_TXT_ENC_UTF8

LV_TXT_ENC_ASCII

LV_TEXT_LEN_MAX

Enums

enum **lv_text_flag_t**

Options for text rendering.

Values:

enumerator **LV_TEXT_FLAG_NONE**

enumerator **LV_TEXT_FLAG_EXPAND**

Ignore max-width to avoid automatic word wrapping

enumerator **LV_TEXT_FLAG_FIT**

Max-width is already equal to the longest line. (Used to skip some calculation)

enumerator **LV_TEXT_FLAG_BREAK_ALL**

To prevent overflow, insert breaks between any two characters. Otherwise breaks are inserted at word boundaries, as configured via LV_TXT_BREAK_CHARS or according to LV_TXT_LINE_BREAK_LONG_LEN, LV_TXT_LINE_BREAK_LONG_PRE_MIN_LEN, and LV_TXT_LINE_BREAK_LONG_POST_MIN_LEN.

enumerator **LV_TEXT_FLAG_RECOLOR**

Enable parsing of recolor command

enum **lv_text_align_t**

Label align policy

Values:

enumerator **LV_TEXT_ALIGN_AUTO**

Align text auto

enumerator **LV_TEXT_ALIGN_LEFT**

Align text to left

enumerator **LV_TEXT_ALIGN_CENTER**

Align text to center

enumerator **LV_TEXT_ALIGN_RIGHT**

Align text to right

enum **lv_text_cmd_state_t**

State machine for text renderer.

Values:

enumerator **LV_TEXT_CMD_STATE_WAIT**

Waiting for command

enumerator **LV_TEXT_CMD_STATE_PAR**

Processing the parameter

enumerator **LV_TEXT_CMD_STATE_IN**

Processing the command

Functions

void **lv_text_get_size**(*lv_point_t* *size_res, const char *text, const lv_font_t *font, int32_t letter_space, int32_t line_space, int32_t max_width, *lv_text_flag_t* flag)

Get size of a text

line breaks

Parametri

- **size_res** -- pointer to a 'point_t' variable to store the result
- **text** -- pointer to a text
- **font** -- pointer to font of the text
- **letter_space** -- letter space of the text
- **line_space** -- line space of the text
- **max_width** -- max width of the text (break the lines to fit this size). Set COORD_MAX to avoid
- **flag** -- settings for the text from *lv_text_flag_t*

int32_t **lv_text_get_width**(const char *txt, uint32_t length, const lv_font_t *font, int32_t letter_space)

Give the length of a text with a given font

Parametri

- **txt** -- a '\0' terminate string
- **length** -- length of 'txt' in byte count and not characters (Á is 1 character but 2 bytes in UTF-8)
- **font** -- pointer to a font
- **letter_space** -- letter space

Ritorna

length of a char_num long text

int32_t **lv_text_get_width_with_flags**(const char *txt, uint32_t length, const lv_font_t *font, int32_t letter_space, *lv_text_flag_t* flags)

Give the length of a text with a given font with text flags

Parametri

- **txt** -- a '\0' terminate string
- **length** -- length of 'txt' in byte count and not characters (Á is 1 character but 2 bytes in UTF-8)
- **font** -- pointer to a font
- **letter_space** -- letter space
- **flags** -- settings for the text from *lv_text_flag_t*

Ritorna

length of a char_num long text

bool **lv_text_is_cmd**(*lv_text_cmd_state_t* *state, uint32_t c)

Check if c is command state

Parametri

- **state** --
- **c** --

Ritorna

True if c is state

14.18.37 lv_text_ap.h

Defines

LV_UNDEF_ARABIC_PERSIAN_CHARS

LV_AP_ALPHABET_BASE_CODE

LV_AP_END_CHARS_LIST

Functions

uint32_t **lv_text_ap_calc_bytes_count**(const char *txt)

void **lv_text_ap_proc**(const char *txt, char *txt_out)

14.18.38 lv_text_private.h

Functions

uint32_t **lv_text_get_next_line**(const char *txt, uint32_t len, const lv_font_t *font, int32_t letter_space, int32_t max_width, int32_t *used_width, *lv_text_flag_t* flag)

Get the next line of text. Check line length and break chars too.

Parametri

- **txt** -- a '\0' terminated string
- **len** -- length of 'txt' in bytes
- **font** -- pointer to a font
- **letter_space** -- letter space
- **max_width** -- max width of the text (break the lines to fit this size). Set COORD_MAX to avoid line breaks
- **used_width** -- When used_width != NULL, save the width of this line if flag == LV_TEXT_FLAG_NONE, otherwise save -1.
- **flag** -- settings for the text from 'txt_flag_type' enum

Ritorna

the index of the first char of the new line (in byte index not letter index. With UTF-8 they are different)

void **lv_text_ins**(char *txt_buf, uint32_t pos, const char *ins_txt)

Insert a string into another

Parametri

- **txt_buf** -- the original text (must be big enough for the result text and NULL terminated)
- **pos** -- position to insert (0: before the original text, 1: after the first char etc.)
- **ins_txt** -- text to insert, must be '\0' terminated

void **lv_text_cut**(char *txt, uint32_t pos, uint32_t len)

Delete a part of a string

Parametri

- **txt** -- string to modify, must be '\0' terminated and should point to a heap or stack frame, not read-only memory.
- **pos** -- position where to start the deleting (0: before the first char, 1: after the first char etc.)
- **len** -- number of characters to delete

char ***lv_text_set_text_vfmt**(const char *fmt, va_list ap)

return a new formatted text. Memory will be allocated to store the text.

Parametri

- **fmt** -- `printf`-like format
- **ap** -- items to print

Ritorna

pointer to the allocated text string.

void **lv_text_encoded_letter_next_2**(const char *txt, uint32_t *letter, uint32_t *letter_next, uint32_t *ofs)

Decode two encoded character from a string.

Parametri

- **txt** -- pointer to '\0' terminated string
- **letter** -- the first decoded Unicode character or 0 on invalid data code
- **letter_next** -- the second decoded Unicode character or 0 on invalid data code

- **ofs** -- start index in 'txt' where to start. After the call it will point to the next encoded char in 'txt'. NULL to use txt[0] as index

static inline bool **lv_text_is_break_char**(uint32_t letter)

Test if char is break char or not (a text can broken here or not)

Parametri

letter -- a letter

Ritorna

false: 'letter' is not break char

static inline bool **lv_text_is_a_word**(uint32_t letter)

Test if char is break char or not (a text can broken here or not)

Parametri

letter -- a letter

Ritorna

false: 'letter' is not break char

static inline bool **lv_text_is_marker**(uint32_t letter)

Test if character can be treated as marker, and don't need to be rendered. Note, this is not a full list. Add your findings to the list.

Parametri

letter -- a letter

Ritorna

true if so

Variables

uint8_t (*const **lv_text_encoded_size**)(const char *txt)

Give the size of an encoded character

Param txt

pointer to a character in a string

Return

length of the encoded character (1,2,3 ...). O in invalid

uint32_t (*const **lv_text_unicode_to_encoded**)(uint32_t letter_uni)

Convert a Unicode letter to encoded

Param letter_uni

a Unicode letter

Return

Encoded character in Little Endian to be compatible with C chars (e.g. 'Á', 'Ü')

uint32_t (*const **lv_text_encoded_conv_wc**)(uint32_t c)

Convert a wide character, e.g. 'Á' little endian to be compatible with the encoded format.

Param c

a wide character

Return

`c` in the encoded format

uint32_t (*const **lv_text_encoded_next**)(const char *txt, uint32_t *i_start)

Decode the next encoded character from a string.

Param txt

pointer to '\0' terminated string

Param i_start

start index in 'txt' where to start. After the call it will point to the next encoded char in 'txt'. NULL to use txt[0] as index

Return

the decoded Unicode character or 0 on invalid data code

uint32_t (*const **lv_text_encoded_prev**)(const char *txt, uint32_t *i_start)

Get the previous encoded character form a string.

Param txt
pointer to '\0' terminated string

Param i_start
index in 'txt' where to start. After the call it will point to the previous encoded char in 'txt'.

Return
the decoded Unicode character or 0 on invalid data

uint32_t (*const **lv_text_encoded_get_byte_id**)(const char *txt, uint32_t utf8_id)

Convert a letter index (in the encoded text) to byte index. E.g. in UTF-8 "AÁRT" index of 'R' is 2 but start at byte 3 because 'Á' is 2 bytes long

Param txt
a '\0' terminated UTF-8 string

Param utf8_id
character index

Return
byte index of the 'enc_id'th letter

uint32_t (*const **lv_text_encoded_get_char_id**)(const char *txt, uint32_t byte_id)

Convert a byte index (in an encoded text) to character index. E.g. in UTF-8 "AÁRT" index of 'R' is 2 but start at byte 3 because 'Á' is 2 bytes long

Param txt
a '\0' terminated UTF-8 string

Param byte_id
byte index

Return
character index of the letter at 'byte_id'th position

uint32_t (*const **lv_text_get_encoded_length**)(const char *txt)

Get the number of characters (and NOT bytes) in a string. E.g. in UTF-8 "ÁBC" is 3 characters (but 4 bytes)

Param txt
a '\0' terminated char string

Return
number of characters

14.18.39 lv_timer.h

Defines

LV_NO_TIMER_READY

Typedef

typedef void (***lv_timer_cb_t**)(lv_timer_t*)

Timers execute this type of functions.

typedef void (***lv_timer_handler_resume_cb_t**)(void *data)

Timer handler resume this type of function.

Functions

uint32_t **lv_timer_handler**(void)

Call it periodically to handle lv_timers.

Ritorna

time till it needs to be run next (in ms)

uint32_t **lv_timer_handler_run_in_period**(uint32_t period)

Call it in the super-loop of main() or threads. It will run *lv_timer_handler()* with a given period in ms. You can use it with sleep or delay in OS environment. This function is used to simplify the porting.

Parametri

period -- the period for running *lv_timer_handler()*

Ritorna

the time after which it must be called again

void **lv_timer_periodic_handler**(void)

Call it in the super-loop of main() or threads. It will automatically call *lv_timer_handler()* at the right time. This function is used to simplify the porting.

void **lv_timer_handler_set_resume_cb**(*lv_timer_handler_resume_cb_t* cb, void *data)

Set the resume callback to the timer handler

Parametri

- **cb** -- the function to call when timer handler is resumed
- **data** -- pointer to a resume data

lv_timer_t ***lv_timer_create_basic**(void)

Create an "empty" timer. It needs to be initialized with at least `lv_timer_set_cb` and `lv_timer_set_period`

Ritorna

pointer to the created timer

lv_timer_t ***lv_timer_create**(*lv_timer_cb_t* timer_xcb, uint32_t period, void *user_data)

Create a new lv_timer

Parametri

- **timer_xcb** -- a callback to call periodically. (the 'x' in the argument name indicates that it's not a fully generic function because it not follows the `func_name(object, callback, ...)` convention)
- **period** -- call period in ms unit
- **user_data** -- custom parameter

Ritorna

pointer to the new timer

void **lv_timer_delete**(lv_timer_t *timer)

Delete a lv_timer

Parametri

timer -- pointer to an lv_timer

void **lv_timer_pause**(lv_timer_t *timer)

Pause a timer.

Parametri

timer -- pointer to an lv_timer

void **lv_timer_resume**(lv_timer_t *timer)

Resume a timer.

Parametri

timer -- pointer to an lv_timer

void **lv_timer_set_cb**(lv_timer_t *timer, *lv_timer_cb_t* timer_cb)

Set the callback to the timer (the function to call periodically)

Parametri

- **timer** -- pointer to a timer

- **timer_cb** -- the function to call periodically

void **lv_timer_set_period**(lv_timer_t *timer, uint32_t period)

Set new period for a lv_timer

Parametri

- **timer** -- pointer to a lv_timer
- **period** -- the new period

void **lv_timer_ready**(lv_timer_t *timer)

Make a lv_timer ready. It will not wait its period.

Parametri

timer -- pointer to a lv_timer.

void **lv_timer_set_repeat_count**(lv_timer_t *timer, int32_t repeat_count)

Set the number of times a timer will repeat.

Parametri

- **timer** -- pointer to a lv_timer.
- **repeat_count** -- -1 : infinity; 0 : stop ; n>0: residual times

void **lv_timer_set_auto_delete**(lv_timer_t *timer, bool auto_delete)

Set whether a lv_timer will be deleted automatically when it is called `repeat_count` times.

Parametri

- **timer** -- pointer to a lv_timer.
- **auto_delete** -- true: auto delete; false: timer will be paused when it is called `repeat_count` times.

void **lv_timer_set_user_data**(lv_timer_t *timer, void *user_data)

Set custom parameter to the lv_timer.

Parametri

- **timer** -- pointer to a lv_timer.
- **user_data** -- custom parameter

void **lv_timer_reset**(lv_timer_t *timer)

Reset a lv_timer. It will be called the previously set period milliseconds later.

Parametri

timer -- pointer to a lv_timer.

void **lv_timer_enable**(bool en)

Enable or disable the whole lv_timer handling

Parametri

en -- true: lv_timer handling is running, false: lv_timer handling is suspended

uint32_t **lv_timer_get_idle**(void)

Get idle percentage

Ritorna

the lv_timer idle in percentage

uint32_t **lv_timer_get_time_until_next**(void)

Get the time remaining until the next timer will run

Ritorna

the time remaining in ms

lv_timer_t ***lv_timer_get_next**(lv_timer_t *timer)

Iterate through the timers

Parametri

timer -- NULL to start iteration or the previous return value to get the next timer

Ritorna

the next timer or NULL if there is no more timer

void ***lv_timer_get_user_data**(lv_timer_t *timer)

Get the user_data passed when the timer was created

Parametri
timer -- pointer to the lv_timer
Ritorna
pointer to the user_data

bool **lv_timer_get_paused**(lv_timer_t *timer)

Get the pause state of a timer

Parametri
timer -- pointer to a lv_timer
Ritorna
true: timer is paused; false: timer is running

14.18.40 lv_timer_private.h

Functions

void **lv_timer_core_init**(void)

Init the lv_timer module

void **lv_timer_core_deinit**(void)

Deinit the lv_timer module

struct **_lv_timer_t**

#include <lv_timer_private.h> Descriptor of a lv_timer

Public Members

uint32_t **period**

How often the timer should run

uint32_t **last_run**

Last time the timer ran

lv_timer_cb_t **timer_cb**

Timer function

void ***user_data**

Custom user data

int32_t **repeat_count**

1: One time; -1 : infinity; n>0: residual times

uint32_t **paused**

uint32_t **auto_delete**

struct **lv_timer_state_t**

Public Members

lv_ll_t **timer_ll**

Linked list to store the lv_timers

bool **lv_timer_run**

uint8_t **idle_last**

bool **timer_deleted**

bool **timer_created**

uint32_t **timer_time_until_next**

bool **already_running**

uint32_t **periodic_last_tick**

uint32_t **busy_time**

uint32_t **idle_period_start**

uint32_t **run_cnt**

lv_timer_handler_resume_cb_t **resume_cb**

void ***resume_data**

14.18.41 lv_tree.h

Tree. The tree nodes are dynamically allocated by the 'lv_mem' module.

Defines

LV_TREE_NODE(n)

Typedef

typedef struct *_lv_tree_class_t* **lv_tree_class_t**

typedef struct *_lv_tree_node_t* **lv_tree_node_t**

typedef void (***lv_tree_constructor_cb_t**)(const *lv_tree_class_t* *class_p, *lv_tree_node_t* *node)

typedef void (***lv_tree_destructor_cb_t**)(const *lv_tree_class_t* *class_p, *lv_tree_node_t* *node)

typedef uint8_t **lv_tree_walk_mode_t**

typedef bool (***lv_tree_traverse_cb_t**)(const *lv_tree_node_t* *node, void *user_data)

typedef bool (***lv_tree_before_cb_t**)(const *lv_tree_node_t* *node, void *user_data)

typedef void (*`lv_tree_after_cb_t`)(const *lv_tree_node_t* *node, void *user_data)

Enums

enum `_lv_tree_walk_mode_t`

Values:

enumerator `LV_TREE_WALK_PRE_ORDER`

enumerator `LV_TREE_WALK_POST_ORDER`

Functions

lv_tree_node_t *`lv_tree_node_create`(const *lv_tree_class_t* *class_p, *lv_tree_node_t* *parent)

Create a tree node.

Parametri

- `class_p` -- pointer to a class of the node
- `parent` -- pointer to the parent node (or NULL if it's the root node)

Ritorna

pointer to the new node

void `lv_tree_node_delete`(*lv_tree_node_t* *node)

Delete a tree node and all its children recursively.

Parametri

`node` -- pointer to the node to delete

bool `lv_tree_walk`(const *lv_tree_node_t* *node, *lv_tree_walk_mode_t* mode, *lv_tree_traverse_cb_t* cb, *lv_tree_before_cb_t* bcb, *lv_tree_after_cb_t* acb, void *user_data)

Walk the tree recursively and call a callback function on each node.

Parametri

- `node` -- pointer to the root node of the tree
- `mode` -- LV_TREE_WALK_PRE_ORDER or LV_TREE_WALK_POST_ORDER
- `cb` -- callback function to call on each node
- `bcb` -- callback function to call before visiting a node
- `acb` -- callback function to call after visiting a node
- `user_data` -- user data to pass to the callback functions

Ritorna

true: traversal is finished; false: traversal broken

Variables

const *lv_tree_class_t* `lv_tree_node_class`

struct `_lv_tree_class_t`

#include <lv_tree.h> Describe the common methods of every object. Similar to a C++ class.

Public Members

const *lv_tree_class_t* *`base_class`

uint32_t `instance_size`

lv_tree_constructor_cb_t **constructor_cb**

lv_tree_destructor_cb_t **destructor_cb**

struct **_lv_tree_node_t**

#include <lv_tree.h> Description of a tree node

Public Members

lv_tree_node_t ***parent**

lv_tree_node_t ****children**

uint32_t **child_cnt**

uint32_t **child_cap**

const *lv_tree_class_t* ***class_p**

14.18.42 lv_utils.h

Functions

void ***lv_utils_bsearch**(const void *key, const void *base, size_t n, size_t size, int (*cmp)(const void *pRef, const void *pElement))

Searches base[0] to base[n - 1] for an item that matches *key.

> **Nota**
>
> The function cmp must return negative if it's first argument (the search key) is less that it's second (a table entry), zero if equal, and positive if greater.

> **Nota**
>
> Items in the array must be in ascending order.

Parametri

- **key** -- Pointer to item being searched for
- **base** -- Pointer to first element to search
- **n** -- Number of elements
- **size** -- Size of each element
- **cmp** -- Pointer to comparison function (see unicode_list_compare() as a comparison function example)

Ritorna

a pointer to a matching item, or NULL if none exists.

lv_result_t **lv_draw_buf_save_to_file**(const lv_draw_buf_t *draw_buf, const char *path)

Save a draw buf to a file

Parametri

- **draw_buf** -- pointer to a draw buffer
- **path** -- path to the file to save

Ritorna
LV_RESULT_OK: success; LV_RESULT_INVALID: error

static inline uint32_t **lv_swap_bytes_32**(uint32_t x)

Reverse the order of the bytes in a 32-bit value.

Parametri
x -- a 32-bit value.

Ritorna
the value x with reversed byte-order.

static inline uint16_t **lv_swap_bytes_16**(uint16_t x)

Reverse the order of the bytes in a 16-bit value.

Parametri
x -- a 16-bit value.

Ritorna
the value x with reversed byte-order.

14.18.43 cache

lv_cache.h

Functions

lv_cache_t ***lv_cache_create**(const *lv_cache_class_t* *cache_class, size_t node_size, size_t max_size, *lv_cache_ops_t* ops)

Create a cache object with the given parameters.

Parametri
- **cache_class** -- The class of the cache. Currently only support one two builtin classes:
  - lv_cache_class_lru_rb_count for LRU-based cache with count-based eviction policy.
  - lv_cache_class_lru_rb_size for LRU-based cache with size-based eviction policy.
- **node_size** -- The node size is the size of the data stored in the cache..
- **max_size** -- The max size is the maximum amount of memory or count that the cache can hold.
  - lv_cache_class_lru_rb_count: max_size is the maximum count of nodes in the cache.
  - lv_cache_class_lru_rb_size: max_size is the maximum size of the cache in bytes.
- **ops** -- A set of operations that can be performed on the cache. See lv_cache_ops_t for details.

Ritorna
Returns a pointer to the created cache object on success, NULL on error.

void **lv_cache_destroy**(lv_cache_t *cache, void *user_data)

Destroy a cache object.

Parametri
- **cache** -- The cache object pointer to destroy.
- **user_data** -- A user data pointer that will be passed to the free callback.

lv_cache_entry_t ***lv_cache_acquire**(lv_cache_t *cache, const void *key, void *user_data)

Acquire a cache entry with the given key. If entry not in cache, it will return NULL (not found). If the entry is found, it's priority will be changed by the cache's policy. And the lv_cache_entry_t::ref_cnt will be incremented.

Parametri
- **cache** -- The cache object pointer to acquire the entry.
- **key** -- The key of the entry to acquire.

- **user_data** -- A user data pointer that will be passed to the create callback.

Ritorna

Returns a pointer to the acquired cache entry on success with `lv_cache_entry_t::ref_cnt` incremented, `NULL` on error.

lv_cache_entry_t ***lv_cache_acquire_or_create**(lv_cache_t *cache, const void *key, void *user_data)

Acquire a cache entry with the given key. If the entry is not in the cache, it will create a new entry with the given key. If the entry is found, it's priority will be changed by the cache's policy. And the `lv_cache_entry_t::ref_cnt` will be incremented. If you want to use this API to simplify the code, you should provide a *`lv_cache_ops_t::create_cb`* that creates a new entry with the given key. This API is a combination of *lv_cache_acquire()* and *lv_cache_add()*. The effect is the same as calling *lv_cache_acquire()* and *lv_cache_add()* separately. And the internal impact on cache is also consistent with these two APIs.

Parametri

- **cache** -- The cache object pointer to acquire the entry.
- **key** -- The key of the entry to acquire or create.
- **user_data** -- A user data pointer that will be passed to the create callback.

Ritorna

Returns a pointer to the acquired or created cache entry on success with `lv_cache_entry_t::ref_cnt` incremented, `NULL` on error.

lv_cache_entry_t ***lv_cache_add**(lv_cache_t *cache, const void *key, void *user_data)

Add a new cache entry with the given key and data. If the cache is full, the cache's policy will be used to evict an entry.

Parametri

- **cache** -- The cache object pointer to add the entry.
- **key** -- The key of the entry to add.
- **user_data** -- A user data pointer that will be passed to the create callback.

Ritorna

Returns a pointer to the added cache entry on success with `lv_cache_entry_t::ref_cnt` incremented, `NULL` on error.

void **lv_cache_release**(lv_cache_t *cache, lv_cache_entry_t *entry, void *user_data)

Release a cache entry. The `lv_cache_entry_t::ref_cnt` will be decremented. If the `lv_cache_entry_t::ref_cnt` is zero, it will issue an error. If the entry passed to this function is the last reference to the data and the entry is marked as invalid, the cache's policy will be used to evict the entry.

Parametri

- **cache** -- The cache object pointer to release the entry.
- **entry** -- The cache entry pointer to release.
- **user_data** -- A user data pointer that will be passed to the free callback.

void **lv_cache_reserve**(lv_cache_t *cache, uint32_t reserved_size, void *user_data)

Reserve a certain amount of memory/count in the cache. This function is useful when you want to reserve a certain amount of memory/count in advance, for example, when you know that you will need it later. When the current cache size is max than the reserved size, the function will evict entries until the reserved size is reached.

Parametri

- **cache** -- The cache object pointer to reserve.
- **reserved_size** -- The amount of memory/count to reserve.
- **user_data** -- A user data pointer that will be passed to the free callback.

void **lv_cache_drop**(lv_cache_t *cache, const void *key, void *user_data)

Drop a cache entry with the given key. If the entry is not in the cache, nothing will happen to it. If the entry is found, it will be removed from the cache and its data will be freed when the last reference to it is released.

Nota

The data will not be freed immediately but when the last reference to it is released. But this entry will not be found by *lv_cache_acquire()*. If you want cache a same key again, you should use *lv_cache_add()* or *lv_cache_acquire_or_create()*.

Parametri

- **cache** -- The cache object pointer to drop the entry.
- **key** -- The key of the entry to drop.
- **user_data** -- A user data pointer that will be passed to the free callback.

void **lv_cache_drop_all**(lv_cache_t *cache, void *user_data)

Drop all cache entries. All entries will be removed from the cache and their data will be freed when the last reference to them is released.

Nota

If some entries are still referenced by other objects, it will issue an error. And this case shouldn't happen in normal cases..

Parametri

- **cache** -- The cache object pointer to drop all entries.
- **user_data** -- A user data pointer that will be passed to the free callback.

bool **lv_cache_evict_one**(lv_cache_t *cache, void *user_data)

Evict one entry from the cache. The eviction policy will be used to select the entry to evict.

Parametri

- **cache** -- The cache object pointer to evict an entry.
- **user_data** -- A user data pointer that will be passed to the free callback.

Ritorna

Returns true if an entry is evicted, false if no entry is evicted.

void **lv_cache_set_max_size**(lv_cache_t *cache, size_t max_size, void *user_data)

Set the maximum size of the cache. If the current cache size is greater than the new maximum size, the cache's policy will be used to evict entries until the new maximum size is reached. If set to 0, the cache will be disabled.

Nota

But this behavior will happen only new entries are added to the cache.

Parametri

- **cache** -- The cache object pointer to set the maximum size.
- **max_size** -- The new maximum size of the cache.
- **user_data** -- A user data pointer that will be passed to the free callback.

size_t **lv_cache_get_max_size**(lv_cache_t *cache, void *user_data)

Get the maximum size of the cache.

Parametri

- **cache** -- The cache object pointer to get the maximum size.
- **user_data** -- A user data pointer that will be passed to the free callback.

Ritorna

Returns the maximum size of the cache.

size_t **lv_cache_get_size**(lv_cache_t *cache, void *user_data)

Get the current size of the cache.

Parametri

- **cache** -- The cache object pointer to get the current size.
- **user_data** -- A user data pointer that will be passed to the free callback.

Ritorna

Returns the current size of the cache.

size_t **lv_cache_get_free_size**(lv_cache_t *cache, void *user_data)

Get the free size of the cache.

Parametri

- **cache** -- The cache object pointer to get the free size.
- **user_data** -- A user data pointer that will be passed to the free callback.

Ritorna

Returns the free size of the cache.

bool **lv_cache_is_enabled**(lv_cache_t *cache)

Return true if the cache is enabled. Disabled cache means that when the max_size of the cache is 0. In this case, all cache operations will be no-op.

Parametri

cache -- The cache object pointer to check if it's disabled.

Ritorna

Returns true if the cache is enabled, false otherwise.

void **lv_cache_set_compare_cb**(lv_cache_t *cache, *lv_cache_compare_cb_t* compare_cb, void *user_data)

Set the compare callback of the cache.

Parametri

- **cache** -- The cache object pointer to set the compare callback.
- **compare_cb** -- The compare callback to set.
- **user_data** -- A user data pointer.

void **lv_cache_set_create_cb**(lv_cache_t *cache, *lv_cache_create_cb_t* alloc_cb, void *user_data)

Set the create callback of the cache.

Parametri

- **cache** -- The cache object pointer to set the create callback.
- **alloc_cb** -- The create callback to set.
- **user_data** -- A user data pointer.

void **lv_cache_set_free_cb**(lv_cache_t *cache, *lv_cache_free_cb_t* free_cb, void *user_data)

Set the free callback of the cache.

Parametri

- **cache** -- The cache object pointer to set the free callback.
- **free_cb** -- The free callback to set.
- **user_data** -- A user data pointer.

void **lv_cache_set_name**(lv_cache_t *cache, const char *name)

Give a name for a cache object. Only the pointer of the string is saved.

Parametri

- **cache** -- The cache object pointer to set the name.
- **name** -- The name of the cache.

const char ***lv_cache_get_name**(lv_cache_t *cache)

Get the name of a cache object.

Parametri

cache -- The cache object pointer to get the name.

Ritorna

Returns the name of the cache.

lv_iter_t ***lv_cache_iter_create**(lv_cache_t *cache)

Create an iterator for the cache object. The iterator is used to iterate over all cache entries.

Parametri

cache -- The cache object pointer to create the iterator.

Ritorna

Returns a pointer to the created iterator on success, `NULL` on error.

lv_cache_entry.h

Functions

uint32_t **lv_cache_entry_get_size**(const uint32_t node_size)

Get the size of a cache entry.

Parametri

node_size -- The size of the node in the cache.

Ritorna

The size of the cache entry.

int32_t **lv_cache_entry_get_ref**(lv_cache_entry_t *entry)

Get the reference count of a cache entry.

Parametri

entry -- The cache entry to get the reference count of.

Ritorna

The reference count of the cache entry.

uint32_t **lv_cache_entry_get_node_size**(lv_cache_entry_t *entry)

Get the node size of a cache entry. Which is the same size with *lv_cache_entry_get_size()*'s node_size parameter.

Parametri

entry -- The cache entry to get the node size of.

Ritorna

The node size of the cache entry.

bool **lv_cache_entry_is_invalid**(lv_cache_entry_t *entry)

Check if a cache entry is invalid.

Parametri

entry -- The cache entry to check.

Ritorna

True: the cache entry is invalid. False: the cache entry is valid.

void ***lv_cache_entry_get_data**(lv_cache_entry_t *entry)

Get the data of a cache entry.

Parametri

entry -- The cache entry to get the data of.

Ritorna

The pointer to the data of the cache entry.

const lv_cache_t ***lv_cache_entry_get_cache**(const lv_cache_entry_t *entry)

Get the cache instance of a cache entry.

Parametri

entry -- The cache entry to get the cache instance of.

Ritorna

The pointer to the cache instance of the cache entry.

lv_cache_entry_t ***lv_cache_entry_get_entry**(void *data, const uint32_t node_size)

Get the cache entry of a data. The data should be allocated by the cache instance.

Parametri

- **data** -- The data to get the cache entry of.
- **node_size** -- The size of the node in the cache.

Ritorna

The pointer to the cache entry of the data.

lv_cache_entry_t ***lv_cache_entry_alloc**(const uint32_t node_size, const lv_cache_t *cache)

Allocate a cache entry.

Parametri

- **node_size** -- The size of the node in the cache.
- **cache** -- The cache instance to allocate the cache entry from.

Ritorna
The pointer to the allocated cache entry.

void **lv_cache_entry_init**(lv_cache_entry_t *entry, const lv_cache_t *cache, const uint32_t node_size)

Initialize a cache entry.

Parametri
- **entry** -- The cache entry to initialize.
- **cache** -- The cache instance to allocate the cache entry from.
- **node_size** -- The size of the node in the cache.

void **lv_cache_entry_delete**(lv_cache_entry_t *entry)

Deallocate a cache entry. And the data of the cache entry will be freed.

Parametri
entry -- The cache entry to deallocate.

lv_cache_entry_private.h

Defines

LV_CACHE_ENTRY_FLAG_INVALID

LV_CACHE_ENTRY_FLAG_DISABLE_DELETE

LV_CACHE_ENTRY_FLAG_CLASS_CUSTOM

Functions

void **lv_cache_entry_reset_ref**(lv_cache_entry_t *entry)

void **lv_cache_entry_inc_ref**(lv_cache_entry_t *entry)

void **lv_cache_entry_dec_ref**(lv_cache_entry_t *entry)

void **lv_cache_entry_set_node_size**(lv_cache_entry_t *entry, uint32_t node_size)

void **lv_cache_entry_set_cache**(lv_cache_entry_t *entry, const lv_cache_t *cache)

void ***lv_cache_entry_acquire_data**(lv_cache_entry_t *entry)

void **lv_cache_entry_release_data**(lv_cache_entry_t *entry, void *user_data)

void **lv_cache_entry_set_flag**(lv_cache_entry_t *entry, uint8_t flags)

void **lv_cache_entry_remove_flag**(lv_cache_entry_t *entry, uint8_t flags)

bool **lv_cache_entry_has_flag**(lv_cache_entry_t *entry, uint8_t flags)

struct **_lv_cache_entry_t**

Public Members

const lv_cache_t ***cache**

int32_t **ref_cnt**

uint32_t **node_size**

uint8_t **flags**

lv_cache_private.h

Typedef

typedef struct *_lv_cache_ops_t* **lv_cache_ops_t**

typedef struct *_lv_cache_class_t* **lv_cache_class_t**

typedef int8_t **lv_cache_compare_res_t**

typedef bool (*__lv_cache_create_cb_t__)(void *node, void *user_data)

typedef void (*__lv_cache_free_cb_t__)(void *node, void *user_data)

typedef *lv_cache_compare_res_t* (*__lv_cache_compare_cb_t__)(const void *a, const void *b)

typedef void *(*__lv_cache_alloc_cb_t__)(void)

The cache instance allocation function, used by the cache class to allocate memory for cache instances.

Return

It should return a pointer to the allocated instance.

typedef bool (*__lv_cache_init_cb_t__)(lv_cache_t *cache)

The cache instance initialization function, used by the cache class to initialize the cache instance.

Return

It should return true if the initialization is successful, false otherwise.

typedef void (*__lv_cache_destroy_cb_t__)(lv_cache_t *cache, void *user_data)

The cache instance destruction function, used by the cache class to destroy the cache instance.

typedef lv_cache_entry_t *(*__lv_cache_get_cb_t__)(lv_cache_t *cache, const void *key, void *user_data)

The cache get function, used by the cache class to get a cache entry by its key.

Return

`NULL` if the key is not found.

typedef lv_cache_entry_t *(*__lv_cache_add_cb_t__)(lv_cache_t *cache, const void *key, void *user_data)

The cache add function, used by the cache class to add a cache entry with a given key. This function only cares about how to add the entry, it doesn't check if the entry already exists and doesn't care about is it a victim or not.

Return

the added cache entry, or NULL if the entry is not added.

typedef void (*__lv_cache_remove_cb_t__)(lv_cache_t *cache, lv_cache_entry_t *entry, void *user_data)

The cache remove function, used by the cache class to remove a cache entry from the cache but doesn't free the memory.. This function only cares about how to remove the entry, it doesn't care about is it a victim or not.

typedef void (*__lv_cache_drop_cb_t__)(lv_cache_t *cache, const void *key, void *user_data)

The cache drop function, used by the cache class to remove a cache entry from the cache and free the memory.

typedef void (*__lv_cache_drop_all_cb_t__)(lv_cache_t *cache, void *user_data)

The cache drop all function, used by the cache class to remove all cache entries from the cache and free the memory.

typedef lv_cache_entry_t *(*`lv_cache_get_victim_cb`)(lv_cache_t *cache, void *user_data)

The cache get victim function, used by the cache class to get a victim entry to be evicted.

typedef *lv_cache_reserve_cond_res_t* (*`lv_cache_reserve_cond_cb`)(lv_cache_t *cache, const void *key, size_t size, void *user_data)

The cache reserve condition function, used by the cache class to check if a new entry can be added to the cache without exceeding its maximum size. See *lv_cache_reserve_cond_res_t* for the possible results.

typedef lv_iter_t *(*`lv_cache_iter_create_cb`)(lv_cache_t *cache)

The cache iterator creation function, used by the cache class to create an iterator for the cache.

Return

A pointer to the created iterator, or NULL if the iterator cannot be created.

typedef struct *_lv_cache_slot_size_t* `lv_cache_slot_size_t`

Enums

enum `lv_cache_reserve_cond_res_t`

The result of the cache reserve condition callback

Values:

enumerator `LV_CACHE_RESERVE_COND_OK`

The condition is met and no entries need to be evicted

enumerator `LV_CACHE_RESERVE_COND_TOO_LARGE`

The condition is not met and the reserve size is too large

enumerator `LV_CACHE_RESERVE_COND_NEED_VICTIM`

The condition is not met and a victim is needed to be evicted

enumerator `LV_CACHE_RESERVE_COND_ERROR`

An error occurred while checking the condition

struct `_lv_cache_ops_t`

#include <lv_cache_private.h> The cache operations struct

Public Members

lv_cache_compare_cb_t `compare_cb`

Compare function for keys

lv_cache_create_cb_t `create_cb`

Create function for nodes

lv_cache_free_cb_t `free_cb`

Free function for nodes

struct `_lv_cache_t`

#include <lv_cache_private.h> The cache entry struct

Public Members

const *lv_cache_class_t* ***clz**

Cache class. There are two built-in classes:

- lv_cache_class_lru_rb_count for LRU-based cache with count-based eviction policy.
- lv_cache_class_lru_rb_size for LRU-based cache with size-based eviction policy.

uint32_t **node_size**

Size of a node

uint32_t **max_size**

Maximum size of the cache

uint32_t **size**

Current size of the cache

lv_cache_ops_t **ops**

Cache operations struct *_lv_cache_ops_t*

lv_mutex_t **lock**

Cache lock used to protect the cache in multithreading environments

const char ***name**

Name of the cache

struct **_lv_cache_class_t**

#include <lv_cache_private.h> Cache class struct for building custom cache classes

Examples:

- lv_cache_class_lru_rb_count for LRU-based cache with count-based eviction policy.
- lv_cache_class_lru_rb_size for LRU-based cache with size-based eviction policy.

Public Members

lv_cache_alloc_cb_t **alloc_cb**

The allocation function for cache entries

lv_cache_init_cb_t **init_cb**

The initialization function for cache entries

lv_cache_destroy_cb_t **destroy_cb**

The destruction function for cache entries

lv_cache_get_cb_t **get_cb**

The get function for cache entries

lv_cache_add_cb_t **add_cb**

The add function for cache entries

lv_cache_remove_cb_t **remove_cb**

The remove function for cache entries

lv_cache_drop_cb_t **drop_cb**

The drop function for cache entries

lv_cache_drop_all_cb_t **drop_all_cb**

The drop all function for cache entries

lv_cache_get_victim_cb **get_victim_cb**

The get victim function for cache entries

lv_cache_reserve_cond_cb **reserve_cond_cb**

The reserve condition function for cache entries

lv_cache_iter_create_cb **iter_create_cb**

The iterator creation function for cache entries

struct **_lv_cache_slot_size_t**

#include <lv_cache_private.h> Cache entry slot struct

To add new fields to the cache entry, add them to a new struct and add it to the first field of the cache data struct. And this one is a size slot for the cache entry.

Public Members

size_t **size**

class

lv_cache_class.h

lv_cache_lru_ll.h

Variables

const *lv_cache_class_t* **lv_cache_class_lru_ll_count**

const *lv_cache_class_t* **lv_cache_class_lru_ll_size**

lv_cache_lru_rb.h

Variables

const *lv_cache_class_t* **lv_cache_class_lru_rb_count**

const *lv_cache_class_t* **lv_cache_class_lru_rb_size**

lv_cache_sc_da.h

Variables

const *lv_cache_class_t* **lv_cache_class_sc_da**

instance

lv_cache_instance.h

lv_image_cache.h

Functions

lv_result_t **lv_image_cache_init**(uint32_t size)

Initialize image cache.

Parametri

size -- size of the cache in bytes.

Ritorna

LV_RESULT_OK: initialization succeeded, LV_RESULT_INVALID: failed.

void **lv_image_cache_resize**(uint32_t new_size, bool evict_now)

Resize image cache. If set to 0, the cache will be disabled.

Parametri

- **new_size** -- new size of the cache in bytes.
- **evict_now** -- true: evict the images should be removed by the eviction policy, false: wait for the next cache cleanup.

void **lv_image_cache_drop**(const void *src)

Invalidate image cache. Use NULL to invalidate all images.

Parametri

src -- pointer to an image source.

bool **lv_image_cache_is_enabled**(void)

Return true if the image cache is enabled.

Ritorna

true: enabled, false: disabled.

lv_iter_t ***lv_image_cache_iter_create**(void)

Create an iterator to iterate over the image cache.

Ritorna

an iterator to iterate over the image cache.

void **lv_image_cache_dump**(void)

Dump the content of the image cache in a human-readable format with cache order.

lv_image_header_cache.h

Functions

lv_result_t **lv_image_header_cache_init**(uint32_t count)

Initialize image header cache.

Parametri

count -- initial size of the cache in count of image headers.

Ritorna

LV_RESULT_OK: initialization succeeded, LV_RESULT_INVALID: failed.

void **lv_image_header_cache_resize**(uint32_t count, bool evict_now)

Resize image header cache. If set to 0, the cache is disabled.

Parametri

- **count** -- new max count of cached image headers.
- **evict_now** -- true: evict the image headers should be removed by the eviction policy, false: wait for the next cache cleanup.

void **lv_image_header_cache_drop**(const void *src)

Invalidate image header cache. Use NULL to invalidate all image headers. It's also automatically called when an image is invalidated.

Parametri
`src` -- pointer to an image source.

bool `lv_image_header_cache_is_enabled`(void)

Return true if the image header cache is enabled.

Ritorna
true: enabled, false: disabled.

lv_iter_t *`lv_image_header_cache_iter_create`(void)

Create an iterator to iterate over the image header cache.

Ritorna
an iterator to iterate over the image header cache.

void `lv_image_header_cache_dump`(void)

Dump the content of the image header cache in a human-readable format with cache order.

14.19 osal

14.19.1 lv_linux_private.h

Defines

`LV_PROC_STAT_PARAMS_LEN`

Functions

uint32_t `lv_os_get_proc_idle_percent`(void)

union `lv_proc_stat_t`

Public Members

uint32_t `user`

uint32_t `nice`

uint32_t `system`

uint32_t `idle`

uint32_t `irq`

uint32_t `softirq`

uint32_t `steal`

struct *lv_proc_stat_t* `fields`

uint32_t `buffer`[7]

14.19.2 lv_os.h

Enums

enum **lv_thread_prio_t**

Values:

enumerator **LV_THREAD_PRIO_LOWEST**

enumerator **LV_THREAD_PRIO_LOW**

enumerator **LV_THREAD_PRIO_MID**

enumerator **LV_THREAD_PRIO_HIGH**

enumerator **LV_THREAD_PRIO_HIGHEST**

Functions

uint32_t **lv_os_get_idle_percent**(void)

Set it for `LV_SYSMON_GET_IDLE` to show the CPU usage

Ritorna

the idle percentage since the last call

static inline lv_result_t **lv_thread_init**(*lv_thread_t* *thread, const char *const name, *lv_thread_prio_t* prio, void (*callback)(void*), size_t stack_size, void *user_data)

static inline lv_result_t **lv_thread_delete**(*lv_thread_t* *thread)

static inline lv_result_t **lv_mutex_init**(*lv_mutex_t* *mutex)

static inline lv_result_t **lv_mutex_lock**(*lv_mutex_t* *mutex)

static inline lv_result_t **lv_mutex_lock_isr**(*lv_mutex_t* *mutex)

static inline lv_result_t **lv_mutex_unlock**(*lv_mutex_t* *mutex)

static inline lv_result_t **lv_mutex_delete**(*lv_mutex_t* *mutex)

static inline lv_result_t **lv_thread_sync_init**(*lv_thread_sync_t* *sync)

static inline lv_result_t **lv_thread_sync_wait**(*lv_thread_sync_t* *sync)

static inline lv_result_t **lv_thread_sync_signal**(*lv_thread_sync_t* *sync)

static inline lv_result_t **lv_thread_sync_signal_isr**(*lv_thread_sync_t* *sync)

static inline lv_result_t **lv_thread_sync_delete**(*lv_thread_sync_t* *sync)

static inline void **lv_lock**(void)

static inline lv_result_t **lv_lock_isr**(void)

static inline void **lv_unlock**(void)

14.19.3 lv_os_none.h

Typedef

typedef int **lv_mutex_t**

typedef int **lv_thread_t**

typedef int **lv_thread_sync_t**

14.19.4 lv_os_private.h

Functions

void **lv_os_init**(void)

Initialize the OS layer

14.20 others

14.20.1 file_explorer

lv_file_explorer.h

Enums

enum **lv_file_explorer_sort_t**

Values:

enumerator **LV_EXPLORER_SORT_NONE**

enumerator **LV_EXPLORER_SORT_KIND**

enum **lv_file_explorer_dir_t**

Values:

enumerator **LV_EXPLORER_HOME_DIR**

enumerator **LV_EXPLORER_MUSIC_DIR**

enumerator **LV_EXPLORER_PICTURES_DIR**

enumerator **LV_EXPLORER_VIDEO_DIR**

enumerator **LV_EXPLORER_DOCS_DIR**

enumerator **LV_EXPLORER_FS_DIR**

Functions

lv_obj_t *`lv_file_explorer_create`(lv_obj_t *parent)

void `lv_file_explorer_set_quick_access_path`(lv_obj_t *obj, *lv_file_explorer_dir_t* dir, const char *path)

Set file_explorer

Parametri

- **obj** -- pointer to a label object
- **dir** -- the dir from 'lv_file_explorer_dir_t' enum.
- **path** -- path

void `lv_file_explorer_set_sort`(lv_obj_t *obj, *lv_file_explorer_sort_t* sort)

Set file_explorer sort

Parametri

- **obj** -- pointer to a file explorer object
- **sort** -- the sort from 'lv_file_explorer_sort_t' enum.

const char *`lv_file_explorer_get_selected_file_name`(const lv_obj_t *obj)

Get file explorer Selected file

Parametri

obj -- pointer to a file explorer object

Ritorna

pointer to the file explorer selected file name

const char *`lv_file_explorer_get_current_path`(const lv_obj_t *obj)

Get file explorer cur path

Parametri

obj -- pointer to a file explorer object

Ritorna

pointer to the file explorer cur path

lv_obj_t *`lv_file_explorer_get_file_table`(lv_obj_t *obj)

Get file explorer file list obj(lv_table)

Parametri

obj -- pointer to a file explorer object

Ritorna

pointer to the file explorer file table obj(lv_table)

lv_obj_t *`lv_file_explorer_get_header`(lv_obj_t *obj)

Get file explorer head area obj

Parametri

obj -- pointer to a file explorer object

Ritorna

pointer to the file explorer head area obj(lv_obj)

lv_obj_t *`lv_file_explorer_get_path_label`(lv_obj_t *obj)

Get file explorer path obj(label)

Parametri

obj -- pointer to a file explorer object

Ritorna

pointer to the file explorer path obj(lv_label)

lv_obj_t *`lv_file_explorer_get_quick_access_area`(lv_obj_t *obj)

Get file explorer head area obj

Parametri

obj -- pointer to a file explorer object

Ritorna

pointer to the file explorer quick access area obj(lv_obj)

lv_obj_t *`lv_file_explorer_get_places_list`(lv_obj_t *obj)

Get file explorer places list obj(lv_list)

Parametri

`obj` -- pointer to a file explorer object

Ritorna

pointer to the file explorer places list obj(lv_list)

lv_obj_t *`lv_file_explorer_get_device_list`(lv_obj_t *obj)

Get file explorer device list obj(lv_list)

Parametri

`obj` -- pointer to a file explorer object

Ritorna

pointer to the file explorer device list obj(lv_list)

lv_file_explorer_sort_t `lv_file_explorer_get_sort`(const lv_obj_t *obj)

Set file_explorer sort

Parametri

`obj` -- pointer to a file explorer object

Ritorna

the current mode from 'lv_file_explorer_sort_t'

void `lv_file_explorer_open_dir`(lv_obj_t *obj, const char *dir)

Open a specified path

Parametri

- `obj` -- pointer to a file explorer object
- `dir` -- pointer to the path

Variables

const lv_obj_class_t `lv_file_explorer_class`

lv_file_explorer_private.h

Enums

enum `lv_file_explorer_file_kind_t`

Values:

enumerator `LV_FILE_EXPLORER_FILE_KIND_DIR`

enumerator `LV_FILE_EXPLORER_FILE_KIND_IMAGE`

enumerator `LV_FILE_EXPLORER_FILE_KIND_AUDIO`

enumerator `LV_FILE_EXPLORER_FILE_KIND_VIDEO`

enumerator `LV_FILE_EXPLORER_FILE_KIND_FILE`

struct `_lv_file_explorer_t`

Public Members

lv_obj_t **obj**

lv_obj_t ***cont**

lv_obj_t ***head_area**

lv_obj_t ***browser_area**

lv_obj_t ***file_table**

lv_obj_t ***path_label**

lv_obj_t ***quick_access_area**

lv_obj_t ***list_device**

lv_obj_t ***list_places**

char ***home_dir**

char ***music_dir**

char ***pictures_dir**

char ***video_dir**

char ***docs_dir**

char ***fs_dir**

const char ***sel_fn**

char **current_path**[(128)]

lv_file_explorer_sort_t **sort**

struct **lv_file_explorer_file_table_entry_data_t**

Public Members

lv_file_explorer_file_kind_t **file_kind**

14.20.2 font_manager

lv_font_manager.h

Functions

lv_font_manager_t *`lv_font_manager_create`(uint32_t recycle_cache_size)

Create main font manager.

Parametri

`recycle_cache_size` -- number of fonts that were recently deleted from the cache.

Ritorna

pointer to main font manager.

bool `lv_font_manager_delete`(lv_font_manager_t *manager)

Delete main font manager.

Parametri

`manager` -- pointer to main font manager.

Ritorna

return true if the deletion was successful.

bool `lv_font_manager_add_src`(lv_font_manager_t *manager, const char *name, const void *src, const lv_font_class_t *class_p)

Add font resource.

Parametri

- `manager` -- pointer to main font manager.
- `name` -- font name.
- `src` -- font source. Need to strictly correspond to the font class.
- `class_p` -- font class. eg. lv_freetype_font_class, lv_builtin_font_class.

Ritorna

return true if the add was successful.

bool `lv_font_manager_add_src_static`(lv_font_manager_t *manager, const char *name, const void *src, const lv_font_class_t *class_p)

Add font resource with static memory.

Parametri

- `manager` -- pointer to main font manager.
- `name` -- font name. It cannot be a local variable.
- `src` -- font source. Need to strictly correspond to the font class. And it cannot be a local variable.
- `class_p` -- font class. E.g. lv_freetype_font_class, lv_builtin_font_class.

Ritorna

return true if the add was successful.

bool `lv_font_manager_remove_src`(lv_font_manager_t *manager, const char *name)

Remove font resource.

Parametri

- `manager` -- pointer to main font manager.
- `name` -- font name.

Ritorna

return true if the remove was successful.

lv_font_t *`lv_font_manager_create_font`(lv_font_manager_t *manager, const char *font_family, uint32_t render_mode, uint32_t size, uint32_t style, *lv_font_kerning_t* kerning)

Create font.

Parametri

- `manager` -- pointer to main font manager.
- `font_family` -- font family name. Matches the font resource name, using commas to separate different names. E.g. "my_font_1,my_font_2".

- **render_mode** -- font render mode. see `lv_freetype_font_render_mode_t`.
- **size** -- font size in pixel.
- **style** -- font style. see `lv_freetype_font_style_t`.
- **kerning** -- kerning mode. see *lv_font_kerning_t*.

Ritorna

point to the created font.

void **lv_font_manager_delete_font**(lv_font_manager_t *manager, lv_font_t *font)

Delete font.

Parametri

- **manager** -- pointer to main font manager.
- **font** -- point to the font.

Ritorna

return true if the deletion was successful.

lv_font_manager_recycle.h

Typedef

typedef struct _lv_font_manager_recycle_t **lv_font_manager_recycle_t**

Functions

lv_font_manager_recycle_t ***lv_font_manager_recycle_create**(uint32_t max_size)

Create font recycle manager.

Parametri

max_size -- recycle size.

Ritorna

pointer to font recycle manager.

void **lv_font_manager_recycle_delete**(*lv_font_manager_recycle_t* *manager)

Delete font recycle manager.

Parametri

manager -- pointer to font recycle manager.

lv_font_t ***lv_font_manager_recycle_get_reuse**(*lv_font_manager_recycle_t* *manager, const lv_font_info_t *ft_info)

Get a reusable font.

Parametri

- **manager** -- pointer to font recycle manager.
- **ft_info** -- font info.

Ritorna

returns true on success.

void **lv_font_manager_recycle_set_reuse**(*lv_font_manager_recycle_t* *manager, lv_font_t *font, const lv_font_info_t *ft_info)

Set fonts to be reused.

Parametri

- **manager** -- pointer to font recycle manager.
- **ft_info** -- font info.

14.20.3 fragment

lv_fragment.h

Public header for Fragment

Typedef

typedef struct _lv_fragment_manager_t **lv_fragment_manager_t**

Functions

lv_fragment_manager_t ***lv_fragment_manager_create**(lv_fragment_t *parent)

Create fragment manager instance

Parametri

parent -- Parent fragment if this manager is placed inside another fragment, can be null.

Ritorna

Fragment manager instance

void **lv_fragment_manager_delete**(*lv_fragment_manager_t* *manager)

Destroy fragment manager instance

Parametri

manager -- Fragment manager instance

void **lv_fragment_manager_create_obj**(*lv_fragment_manager_t* *manager)

Create object of all fragments managed by this manager.

Parametri

manager -- Fragment manager instance

void **lv_fragment_manager_delete_obj**(*lv_fragment_manager_t* *manager)

Delete object created by all fragments managed by this manager. Instance of fragments will not be deleted.

Parametri

manager -- Fragment manager instance

void **lv_fragment_manager_add**(*lv_fragment_manager_t* *manager, lv_fragment_t *fragment, lv_obj_t *const *container)

Attach fragment to manager, and add to container.

Parametri

- **manager** -- Fragment manager instance
- **fragment** -- Fragment instance
- **container** -- Pointer to container object for manager to add objects to

void **lv_fragment_manager_remove**(*lv_fragment_manager_t* *manager, lv_fragment_t *fragment)

Detach and destroy fragment. If fragment is in navigation stack, remove from it.

Parametri

- **manager** -- Fragment manager instance
- **fragment** -- Fragment instance

void **lv_fragment_manager_push**(*lv_fragment_manager_t* *manager, lv_fragment_t *fragment, lv_obj_t *const *container)

Attach fragment to manager and add to navigation stack.

Parametri

- **manager** -- Fragment manager instance
- **fragment** -- Fragment instance
- **container** -- Pointer to container object for manager to add objects to

bool **lv_fragment_manager_pop**(*lv_fragment_manager_t* *manager)

Remove the top-most fragment for stack

Parametri

manager -- Fragment manager instance

Ritorna

true if there is fragment to pop

void **lv_fragment_manager_replace**(*lv_fragment_manager_t* *manager, lv_fragment_t *fragment, lv_obj_t *const *container)

Replace fragment. Old item in the stack will be removed.

Parametri
- **manager** -- Fragment manager instance
- **fragment** -- Fragment instance
- **container** -- Pointer to container object for manager to add objects to

bool **lv_fragment_manager_send_event**(*lv_fragment_manager_t* *manager, int code, void *userdata)

Send event to top-most fragment

Parametri
- **manager** -- Fragment manager instance
- **code** -- User-defined ID of event
- **userdata** -- User-defined data

Ritorna
true if fragment returned true

size_t **lv_fragment_manager_get_stack_size**(*lv_fragment_manager_t* *manager)

Get stack size of this fragment manager

Parametri
manager -- Fragment manager instance

Ritorna
Stack size of this fragment manager

lv_fragment_t ***lv_fragment_manager_get_top**(*lv_fragment_manager_t* *manager)

Get top most fragment instance

Parametri
manager -- Fragment manager instance

Ritorna
Top most fragment instance

lv_fragment_t ***lv_fragment_manager_find_by_container**(*lv_fragment_manager_t* *manager, const lv_obj_t *container)

Find first fragment instance in the container

Parametri
- **manager** -- Fragment manager instance
- **container** -- Container which target fragment added to

Ritorna
First fragment instance in the container

lv_fragment_t ***lv_fragment_manager_get_parent_fragment**(*lv_fragment_manager_t* *manager)

Get parent fragment

Parametri
manager -- Fragment manager instance

Ritorna
Parent fragment instance

lv_fragment_t ***lv_fragment_create**(const lv_fragment_class_t *cls, void *args)

Create a fragment instance.

Parametri
- **cls** -- Fragment class. This fragment must return non null object.
- **args** -- Arguments assigned by fragment manager

Ritorna
Fragment instance

void **lv_fragment_delete**(lv_fragment_t *fragment)

Destroy a fragment.

Parametri
fragment -- Fragment instance.

lv_fragment_manager_t ***lv_fragment_get_manager**(lv_fragment_t *fragment)

Get associated manager of this fragment

Parametri
fragment -- Fragment instance
Ritorna
Fragment manager instance

lv_obj_t *const ***lv_fragment_get_container**(lv_fragment_t *fragment)

Get container object of this fragment
Parametri
fragment -- Fragment instance
Ritorna
Reference to container object

lv_fragment_t ***lv_fragment_get_parent**(lv_fragment_t *fragment)

Get parent fragment of this fragment
Parametri
fragment -- Fragment instance
Ritorna
Parent fragment

lv_obj_t ***lv_fragment_create_obj**(lv_fragment_t *fragment, lv_obj_t *container)

Create object by fragment.
Parametri
- **fragment** -- Fragment instance.
- **container** -- Container of the objects should be created upon.

Ritorna
Created object

void **lv_fragment_delete_obj**(lv_fragment_t *fragment)

Delete created object of a fragment
Parametri
fragment -- Fragment instance.

void **lv_fragment_recreate_obj**(lv_fragment_t *fragment)

Destroy obj in fragment, and recreate them.
Parametri
fragment -- Fragment instance

struct **_lv_fragment_t**

Public Members

const lv_fragment_class_t ***cls**

Class of this fragment

lv_fragment_managed_states_t ***managed**

Managed fragment states. If not null, then this fragment is managed.

> **Avvertimento**
>
> Don't modify values inside this struct!

lv_fragment_manager_t ***child_manager**

Child fragment manager

lv_obj_t ***obj**

lv_obj returned by create_obj_cb

struct **_lv_fragment_class_t**

Public Members

void (***constructor_cb**)(lv_fragment_t *self, void *args)

Constructor function for fragment class

Param self

Fragment instance

Param args

Arguments assigned by fragment manager

void (***destructor_cb**)(lv_fragment_t *self)

Destructor function for fragment class

Param self

Fragment instance, will be freed after this call

void (***attached_cb**)(lv_fragment_t *self)

Fragment attached to manager

Param self

Fragment instance

void (***detached_cb**)(lv_fragment_t *self)

Fragment detached from manager

Param self

Fragment instance

lv_obj_t *(***create_obj_cb**)(lv_fragment_t *self, lv_obj_t *container)

Create objects

Param self

Fragment instance

Param container

Container of the objects should be created upon

Return

Created object, NULL if multiple objects has been created

void (***obj_created_cb**)(lv_fragment_t *self, lv_obj_t *obj)

Param self

Fragment instance

Param obj

lv_obj returned by create_obj_cb

void (***obj_will_delete_cb**)(lv_fragment_t *self, lv_obj_t *obj)

Called before objects in the fragment will be deleted.

Param self

Fragment instance

Param obj

object with this fragment

void (***obj_deleted_cb**)(lv_fragment_t *self, lv_obj_t *obj)

Called when the object created by fragment received `LV_EVENT_DELETE` event

Param self
Fragment instance

Param obj
object with this fragment

bool (*`event_cb`)(lv_fragment_t *self, int code, void *userdata)

Handle event

Param self
Fragment instance

Param which
User-defined ID of event

Param data1
User-defined data

Param data2
User-defined data

size_t `instance_size`

REQUIRED: Allocation size of fragment

lv_fragment_private.h

struct `_lv_fragment_managed_states_t`

#include <lv_fragment_private.h> Fragment states

Public Members

const lv_fragment_class_t *`cls`

Class of the fragment

lv_fragment_manager_t *`manager`

Manager the fragment attached to

lv_obj_t *const *`container`

Container object the fragment adding view to

lv_fragment_t *`instance`

Fragment instance

bool `obj_created`

true between `create_obj_cb` and `obj_deleted_cb`

bool `destroying_obj`

true before `lv_fragment_delete_obj` is called. Don't touch any object if this is true

bool `in_stack`

true if this fragment is in navigation stack that can be popped

14.20.4 gridnav

lv_gridnav.h

Enums

enum **lv_gridnav_ctrl_t**

Values:

enumerator **LV_GRIDNAV_CTRL_NONE**

enumerator **LV_GRIDNAV_CTRL_ROLLOVER**

If there is no next/previous object in a direction, the focus goes to the object in the next/previous row (on left/right keys) or first/last row (on up/down keys)

enumerator **LV_GRIDNAV_CTRL_SCROLL_FIRST**

If an arrow is pressed and the focused object can be scrolled in that direction then it will be scrolled instead of going to the next/previous object. If there is no more room for scrolling the next/previous object will be focused normally

enumerator **LV_GRIDNAV_CTRL_HORIZONTAL_MOVE_ONLY**

Only use left/right keys for grid navigation. Up/down key events will be sent to the focused object.

enumerator **LV_GRIDNAV_CTRL_VERTICAL_MOVE_ONLY**

Only use up/down keys for grid navigation. Left/right key events will be sent to the focused object.

Functions

void **lv_gridnav_add**(lv_obj_t *obj, *lv_gridnav_ctrl_t* ctrl)

Add grid navigation feature to an object. It expects the children to be arranged into a grid-like layout. Although it's not required to have pixel perfect alignment. This feature makes possible to use keys to navigate among the children and focus them. The keys other than arrows and press/release related events are forwarded to the focused child.

Parametri

- **obj** -- pointer to an object on which navigation should be applied.
- **ctrl** -- control flags from *lv_gridnav_ctrl_t*.

void **lv_gridnav_remove**(lv_obj_t *obj)

Remove the grid navigation support from an object

Parametri

obj -- pointer to an object

void **lv_gridnav_set_focused**(lv_obj_t *cont, lv_obj_t *to_focus, *lv_anim_enable_t* anim_en)

Manually focus an object on gridnav container

Parametri

- **cont** -- pointer to a gridnav container
- **to_focus** -- pointer to an object to focus
- **anim_en** -- LV_ANIM_ON/OFF

14.20.5 ime

lv_ime_pinyin.h

Defines

LV_IME_PINYIN_K9_MAX_INPUT

Enums

enum **lv_ime_pinyin_mode_t**

Values:

enumerator **LV_IME_PINYIN_MODE_K26**

enumerator **LV_IME_PINYIN_MODE_K9**

enumerator **LV_IME_PINYIN_MODE_K9_NUMBER**

Functions

lv_obj_t ***lv_ime_pinyin_create**(lv_obj_t *parent)

void **lv_ime_pinyin_set_keyboard**(lv_obj_t *obj, lv_obj_t *kb)

Set the keyboard of Pinyin input method.

Parametri

- **obj** -- pointer to a Pinyin input method object
- **kb** -- pointer to a Pinyin input method keyboard

void **lv_ime_pinyin_set_dict**(lv_obj_t *obj, *lv_pinyin_dict_t* *dict)

Set the dictionary of Pinyin input method.

Parametri

- **obj** -- pointer to a Pinyin input method object
- **dict** -- pointer to a Pinyin input method dictionary

void **lv_ime_pinyin_set_mode**(lv_obj_t *obj, *lv_ime_pinyin_mode_t* mode)

Set mode, 26-key *input(k26)* or 9-key *input(k9)*.

Parametri

- **obj** -- pointer to a Pinyin input method object
- **mode** -- the mode from 'lv_ime_pinyin_mode_t'

lv_obj_t ***lv_ime_pinyin_get_kb**(lv_obj_t *obj)

Set the dictionary of Pinyin input method.

Parametri

obj -- pointer to a Pinyin IME object

Ritorna

pointer to the Pinyin IME keyboard

lv_obj_t ***lv_ime_pinyin_get_cand_panel**(lv_obj_t *obj)

Set the dictionary of Pinyin input method.

Parametri

obj -- pointer to a Pinyin input method object

Ritorna

pointer to the Pinyin input method candidate panel

const *lv_pinyin_dict_t* ***lv_ime_pinyin_get_dict**(lv_obj_t *obj)

Set the dictionary of Pinyin input method.

Parametri

obj -- pointer to a Pinyin input method object

Ritorna

pointer to the Pinyin input method dictionary

Variables

const lv_obj_class_t **lv_ime_pinyin_class**

struct **lv_pinyin_dict_t**

Public Members

const char *const **py**

const char *const **py_mb**

struct **ime_pinyin_k9_py_str_t**

Public Members

char **py_str**[7]

lv_ime_pinyin_private.h

struct **_lv_ime_pinyin_t**

Public Members

lv_obj_t **obj**

lv_obj_t ***kb**

lv_obj_t ***cand_panel**

const *lv_pinyin_dict_t* ***dict**

lv_ll_t **k9_legal_py_ll**

char ***cand_str**

char **input_char**[16]

char **k9_input_str**[7 + 1]

uint16_t **k9_py_ll_pos**

uint16_t **k9_legal_py_count**

uint16_t **k9_input_str_len**

uint16_t **ta_count**

uint16_t **cand_num**

uint16_t **py_page**

uint16_t **py_num**[26]

uint16_t **py_pos**[26]

lv_ime_pinyin_mode_t **mode**

14.20.6 imgfont

lv_imgfont.h

Typedef

typedef const void *(***lv_imgfont_get_path_cb_t**)(const lv_font_t *font, uint32_t unicode, uint32_t unicode_next, int32_t *offset_y, void *user_data)

Functions

lv_font_t ***lv_imgfont_create**(uint16_t height, *lv_imgfont_get_path_cb_t* path_cb, void *user_data)

Creates a image font with info parameter specified.

Parametri

- **height** -- font size
- **path_cb** -- a function to get the image path name of character.
- **user_data** -- pointer to user data

Ritorna

pointer to the new imgfont or NULL if create error.

void **lv_imgfont_destroy**(lv_font_t *font)

Destroy a image font that has been created.

Parametri

font -- pointer to image font handle.

14.20.7 monkey

lv_monkey.h

Typedef

typedef struct _lv_monkey_t **lv_monkey_t**

Functions

void **lv_monkey_config_init**(lv_monkey_config_t *config)

Initialize a monkey config with default values

Parametri

config -- pointer to 'lv_monkey_config_t' variable to initialize

lv_monkey_t ***lv_monkey_create**(const lv_monkey_config_t *config)

Create monkey for test

Parametri

config -- pointer to 'lv_monkey_config_t' variable

Ritorna
pointer to the created monkey

lv_indev_t ***lv_monkey_get_indev**(*lv_monkey_t* *monkey)

Get monkey input device

Parametri
monkey -- pointer to a monkey

Ritorna
pointer to the input device

void **lv_monkey_set_enable**(*lv_monkey_t* *monkey, bool en)

Enable monkey

Parametri
- **monkey** -- pointer to a monkey
- **en** -- set to true to enable

bool **lv_monkey_get_enable**(*lv_monkey_t* *monkey)

Get whether monkey is enabled

Parametri
monkey -- pointer to a monkey

Ritorna
return true if monkey enabled

void **lv_monkey_set_user_data**(*lv_monkey_t* *monkey, void *user_data)

Set the user_data field of the monkey

Parametri
- **monkey** -- pointer to a monkey
- **user_data** -- pointer to the new user_data.

void ***lv_monkey_get_user_data**(*lv_monkey_t* *monkey)

Get the user_data field of the monkey

Parametri
monkey -- pointer to a monkey

Ritorna
the pointer to the user_data of the monkey

void **lv_monkey_delete**(*lv_monkey_t* *monkey)

Delete monkey

Parametri
monkey -- pointer to monkey

struct **lv_range_t**

Public Members

int32_t **min**

int32_t **max**

struct **lv_urange_t**

Public Members

uint32_t **min**

uint32_t **max**

struct **`_lv_monkey_config_t`**

Public Members

lv_indev_type_t **`type`**
Input device type

lv_urange_t **`period_range`**
Monkey execution period

lv_range_t **`input_range`**
The range of input value

lv_monkey_private.h

14.20.8 observer

lv_observer.h

Typedef

typedef void (*`lv_observer_cb_t`)(lv_observer_t *observer, *lv_subject_t* *subject)

Callback called to notify Observer that Subject's value has changed

Param observer
pointer to Observer

Param subject
pointer to Subject being observed

Enums

enum **`lv_subject_type_t`**

Values for *lv_subject_t*'s `type` field

Values:

enumerator **`LV_SUBJECT_TYPE_INVALID`**
indicates Subject not initialized yet

enumerator **`LV_SUBJECT_TYPE_NONE`**
a null value like None or NILt

enumerator **`LV_SUBJECT_TYPE_INT`**
an int32_t

enumerator **`LV_SUBJECT_TYPE_FLOAT`**
a float, requires `LV_USE_FLOAT 1`

enumerator **`LV_SUBJECT_TYPE_POINTER`**
a void pointer

enumerator **`LV_SUBJECT_TYPE_COLOR`**
an *lv_color_t*

enumerator **LV_SUBJECT_TYPE_GROUP**

an array of Subjects

enumerator **LV_SUBJECT_TYPE_STRING**

a char pointer

Functions

void **lv_subject_init_int**(*lv_subject_t* *subject, int32_t value)

Initialize an integer-type Subject.

Parametri

- **subject** -- pointer to Subject
- **value** -- initial value

void **lv_subject_set_int**(*lv_subject_t* *subject, int32_t value)

Set value of an integer Subject and notify Observers.

Parametri

- **subject** -- pointer to Subject
- **value** -- new value

int32_t **lv_subject_get_int**(*lv_subject_t* *subject)

Get current value of an integer Subject.

Parametri

subject -- pointer to Subject

Ritorna

current value

int32_t **lv_subject_get_previous_int**(*lv_subject_t* *subject)

Get previous value of an integer Subject.

Parametri

subject -- pointer to Subject

Ritorna

current value

void **lv_subject_init_float**(*lv_subject_t* *subject, float value)

Initialize an float-type Subject.

Parametri

- **subject** -- pointer to Subject
- **value** -- initial value

void **lv_subject_set_float**(*lv_subject_t* *subject, float value)

Set value of an float Subject and notify Observers.

Parametri

- **subject** -- pointer to Subject
- **value** -- new value

float **lv_subject_get_float**(*lv_subject_t* *subject)

Get current value of an float Subject.

Parametri

subject -- pointer to Subject

Ritorna

current value

float **lv_subject_get_previous_float**(*lv_subject_t* *subject)

Get previous value of an float Subject.

Parametri

subject -- pointer to Subject

Ritorna

current value

void **lv_subject_init_string**(*lv_subject_t* *subject, char *buf, char *prev_buf, size_t size, const char *value)

Initialize a string-type Subject.

> **Nota**
>
> A string Subject stores its own copy of the string, not just the pointer.

Parametri

- **subject** -- pointer to Subject
- **buf** -- pointer to buffer to store string
- **prev_buf** -- pointer to buffer to store previous string; can be NULL if not used
- **size** -- size of buffer(s)
- **value** -- initial value of string, e.g. "hello"

void **lv_subject_copy_string**(*lv_subject_t* *subject, const char *buf)

Copy a string to a Subject and notify Observers if it changed.

Parametri

- **subject** -- pointer to Subject
- **buf** -- new string

void **lv_subject_snprintf**(*lv_subject_t* *subject, const char *format, ...)

Format a new string, updating Subject, and notify Observers if it changed.

Parametri

- **subject** -- pointer to Subject
- **format** -- format string

const char ***lv_subject_get_string**(*lv_subject_t* *subject)

Get current value of a string Subject.

Parametri

subject -- pointer to Subject

Ritorna

pointer to buffer containing current value

const char ***lv_subject_get_previous_string**(*lv_subject_t* *subject)

Get previous value of a string Subject.

> **Nota**
>
> NULL will be returned if NULL was passed in `lv_subject_init_string()` as `prev_buf`.

Parametri

subject -- pointer to Subject

Ritorna

pointer to buffer containing previous value

void **lv_subject_init_pointer**(*lv_subject_t* *subject, void *value)

Initialize a pointer-type Subject.

Parametri

- **subject** -- pointer to Subject
- **value** -- initial value

void **lv_subject_set_pointer**(*lv_subject_t* *subject, void *ptr)

Set value of a pointer Subject and notify Observers (regardless of whether it changed).

Parametri

- **subject** -- pointer to Subject
- **ptr** -- new value

const void *`lv_subject_get_pointer`(*lv_subject_t* *subject)

Get current value of a pointer Subject.

Parametri

`subject` -- pointer to Subject

Ritorna

current value

const void *`lv_subject_get_previous_pointer`(*lv_subject_t* *subject)

Get previous value of a pointer Subject.

Parametri

`subject` -- pointer to Subject

Ritorna

previous value

void `lv_subject_init_color`(*lv_subject_t* *subject, *lv_color_t* color)

Initialize a color-type Subject.

Parametri

- `subject` -- pointer to Subject
- `color` -- initial value

void `lv_subject_set_color`(*lv_subject_t* *subject, *lv_color_t* color)

Set value of a color Subject and notify Observers if it changed.

Parametri

- `subject` -- pointer to Subject
- `color` -- new value

lv_color_t `lv_subject_get_color`(*lv_subject_t* *subject)

Get current value of a color Subject.

Parametri

`subject` -- pointer to Subject

Ritorna

current value

lv_color_t `lv_subject_get_previous_color`(*lv_subject_t* *subject)

Get previous value of a color Subject.

Parametri

`subject` -- pointer to Subject

Ritorna

previous value

void `lv_subject_init_group`(*lv_subject_t* *group_subject, *lv_subject_t* *list[], uint32_t list_len)

Initialize a Group-type Subject.

Parametri

- `group_subject` -- pointer to Group-type Subject
- `list` -- list of other Subject addresses; when any of these have values updated, Observers of `group_subject` will be notified.
- `list_len` -- number of elements in `list[]`

void `lv_subject_deinit`(*lv_subject_t* *subject)

Remove all Observers from a Subject and free allocated memory, and delete any associated Widget-Binding events. This leaves `subject` "disconnected" from all Observers and all associated Widget events established through Widget Binding.

> **Nota**
>
> This can safely be called regardless of whether any Observers added with *`lv_subject_add_observer_obj()`* or bound to a Widget Property with one of the `..._bind_...()` functions.

Parametri
subject -- pointer to Subject

lv_subject_t ***lv_subject_get_group_element**(*lv_subject_t* *subject, int32_t index)

Get an element from Subject Group's list.

Parametri
- **subject** -- pointer to Group-type Subject
- **index** -- index of element to get

Ritorna
pointer to indexed Subject from list, or NULL if index is out of bounds

lv_observer_t ***lv_subject_add_observer**(*lv_subject_t* *subject, *lv_observer_cb_t* observer_cb, void *user_data)

Add Observer to Subject. When Subject's value changes `observer_cb` will be called.

Parametri
- **subject** -- pointer to Subject
- **observer_cb** -- notification callback
- **user_data** -- optional user data

Ritorna
pointer to newly-created Observer

lv_observer_t ***lv_subject_add_observer_obj**(*lv_subject_t* *subject, *lv_observer_cb_t* observer_cb, lv_obj_t *obj, void *user_data)

Add Observer to Subject for a Widget. When the Widget is deleted, Observer will be unsubscribed from Subject automatically.

> **Nota**
>
> Do not call *`lv_observer_remove()`* on Observers created this way. Only clean up such Observers by either:
>
> - deleting the Widget, or
> - calling *`lv_subject_deinit()`* to gracefully de-couple and remove all Observers.

Parametri
- **subject** -- pointer to Subject
- **observer_cb** -- notification callback
- **obj** -- pointer to Widget
- **user_data** -- optional user data

Ritorna
pointer to newly-created Observer

lv_observer_t ***lv_subject_add_observer_with_target**(*lv_subject_t* *subject, *lv_observer_cb_t* observer_cb, void *target, void *user_data)

Add an Observer to a Subject and also save a target pointer.

Parametri
- **subject** -- pointer to Subject
- **observer_cb** -- notification callback
- **target** -- any pointer (NULL is okay)
- **user_data** -- optional user data

Ritorna
pointer to newly-created Observer

void **lv_observer_remove**(lv_observer_t *observer)

Remove Observer from its Subject.

Parametri
observer -- pointer to Observer

void **lv_obj_remove_from_subject**(lv_obj_t *obj, *lv_subject_t* *subject)

Remove Observers associated with Widget `obj` from specified `subject` or all Subjects.

> **Nota**
>
> This function can be used e.g. when a Widget's Subject(s) needs to be replaced by other Subject(s)

Parametri

- **obj** -- pointer to Widget whose Observers should be removed
- **subject** -- Subject to remove Widget from, or NULL to remove from all Subjects

void ***lv_observer_get_target**(lv_observer_t *observer)

Get target of an Observer.

Parametri

observer -- pointer to Observer

Ritorna

pointer to saved target

lv_obj_t ***lv_observer_get_target_obj**(lv_observer_t *observer)

Get target Widget of Observer. This is the same as `lv_observer_get_target()`, except it returns `target` as an `lv_obj_t *`.

Parametri

observer -- pointer to Observer

Ritorna

pointer to saved Widget target

void ***lv_observer_get_user_data**(const lv_observer_t *observer)

Get Observer's user data.

Parametri

observer -- pointer to Observer

Ritorna

void pointer to saved user data

void **lv_subject_notify**(*lv_subject_t* *subject)

Notify all Observers of Subject.

Parametri

subject -- pointer to Subject

void **lv_obj_add_subject_increment_event**(lv_obj_t *obj, *lv_subject_t* *subject, *lv_event_code_t* trigger, int32_t step, int32_t min, int32_t max)

Add an event handler to increment (or decrement) the value of a subject on a trigger.

Parametri

- **obj** -- pointer to a widget
- **subject** -- pointer to a subject to change
- **trigger** -- the trigger on which the subject should be changed
- **step** -- value to add on trigger
- **min** -- the minimum value
- **max** -- the maximum value

void **lv_obj_add_subject_set_int_event**(lv_obj_t *obj, *lv_subject_t* *subject, *lv_event_code_t* trigger, int32_t value)

Set the value of an integer subject.

Parametri

- **obj** -- pointer to a widget
- **subject** -- pointer to a subject to change
- **trigger** -- the trigger on which the subject should be changed
- **value** -- the value to set

void **lv_obj_add_subject_set_float_event**(lv_obj_t *obj, *lv_subject_t* *subject, *lv_event_code_t* trigger, float value)

Set the value of a float subject.

Parametri

- **obj** -- pointer to a widget
- **subject** -- pointer to a subject to change
- **trigger** -- the trigger on which the subject should be changed
- **value** -- the value to set

void **lv_obj_add_subject_set_string_event**(lv_obj_t *obj, *lv_subject_t* *subject, *lv_event_code_t* trigger, const char *value)

Set the value of a string subject.

Parametri

- **obj** -- pointer to a widget
- **subject** -- pointer to a subject to change
- **trigger** -- the trigger on which the subject should be changed
- **value** -- the value to set

lv_observer_t ***lv_obj_bind_style**(lv_obj_t *obj, const *lv_style_t* *style, *lv_style_selector_t* selector, *lv_subject_t* *subject, int32_t ref_value)

Disable a style if a subject's value is not equal to a reference value

Parametri

- **obj** -- pointer to Widget
- **style** -- pointer to a style
- **selector** -- pointer to a selector
- **subject** -- pointer to Subject
- **ref_value** -- reference value to compare Subject's value with

Ritorna

pointer to newly-created Observer

lv_observer_t ***lv_obj_bind_flag_if_eq**(lv_obj_t *obj, *lv_subject_t* *subject, *lv_obj_flag_t* flag, int32_t ref_value)

Set Widget's flag(s) if an integer Subject's value is equal to a reference value, clear flag otherwise.

Parametri

- **obj** -- pointer to Widget
- **subject** -- pointer to Subject
- **flag** -- flag(s) (can be bit-wise OR-ed) to set or clear (e.g. `LV_OBJ_FLAG_HIDDEN`)
- **ref_value** -- reference value to compare Subject's value with

Ritorna

pointer to newly-created Observer

lv_observer_t ***lv_obj_bind_flag_if_not_eq**(lv_obj_t *obj, *lv_subject_t* *subject, *lv_obj_flag_t* flag, int32_t ref_value)

Set Widget's flag(s) if an integer Subject's value is not equal to a reference value, clear flag otherwise.

Parametri

- **obj** -- pointer to Widget
- **subject** -- pointer to Subject
- **flag** -- flag(s) (can be bit-wise OR-ed) to set or clear (e.g. `LV_OBJ_FLAG_HIDDEN`)
- **ref_value** -- reference value to compare Subject's value with

Ritorna

pointer to newly-created Observer

lv_observer_t ***lv_obj_bind_flag_if_gt**(lv_obj_t *obj, *lv_subject_t* *subject, *lv_obj_flag_t* flag, int32_t ref_value)

Set Widget's flag(s) if an integer Subject's value is greater than a reference value, clear flag otherwise.

Parametri

- **obj** -- pointer to Widget

- **subject** -- pointer to Subject
- **flag** -- flag(s) (can be bit-wise OR-ed) to set or clear (e.g. `LV_OBJ_FLAG_HIDDEN`)
- **ref_value** -- reference value to compare Subject's value with

Ritorna

pointer to newly-created Observer

lv_observer_t ***lv_obj_bind_flag_if_ge**(lv_obj_t *obj, *lv_subject_t* *subject, *lv_obj_flag_t* flag, int32_t ref_value)

Set Widget's flag(s) if an integer Subject's value is greater than or equal to a reference value, clear flag otherwise.

Parametri

- **obj** -- pointer to Widget
- **subject** -- pointer to Subject
- **flag** -- flag(s) (can be bit-wise OR-ed) to set or clear (e.g. `LV_OBJ_FLAG_HIDDEN`)
- **ref_value** -- reference value to compare Subject's value with

Ritorna

pointer to newly-created Observer

lv_observer_t ***lv_obj_bind_flag_if_lt**(lv_obj_t *obj, *lv_subject_t* *subject, *lv_obj_flag_t* flag, int32_t ref_value)

Set Widget's flag(s) if an integer Subject's value is less than a reference value, clear flag otherwise.

Parametri

- **obj** -- pointer to Widget
- **subject** -- pointer to Subject
- **flag** -- flag(s) (can be bit-wise OR-ed) to set or clear (e.g. `LV_OBJ_FLAG_HIDDEN`)
- **ref_value** -- reference value to compare Subject's value with

Ritorna

pointer to newly-created Observer

lv_observer_t ***lv_obj_bind_flag_if_le**(lv_obj_t *obj, *lv_subject_t* *subject, *lv_obj_flag_t* flag, int32_t ref_value)

Set Widget's flag(s) if an integer Subject's value is less than or equal to a reference value, clear flag otherwise.

Parametri

- **obj** -- pointer to Widget
- **subject** -- pointer to Subject
- **flag** -- flag(s) (can be bit-wise OR-ed) to set or clear (e.g. `LV_OBJ_FLAG_HIDDEN`)
- **ref_value** -- reference value to compare Subject's value with

Ritorna

pointer to newly-created Observer

lv_observer_t ***lv_obj_bind_state_if_eq**(lv_obj_t *obj, *lv_subject_t* *subject, *lv_state_t* state, int32_t ref_value)

Set Widget's state(s) if an integer Subject's value is equal to a reference value, clear flag otherwise.

Parametri

- **obj** -- pointer to Widget
- **subject** -- pointer to Subject
- **state** -- state(s) (can be bit-wise OR-ed) to set or clear (e.g. `LV_STATE_CHECKED`)
- **ref_value** -- reference value to compare Subject's value with

Ritorna

pointer to newly-created Observer

lv_observer_t ***lv_obj_bind_state_if_not_eq**(lv_obj_t *obj, *lv_subject_t* *subject, *lv_state_t* state, int32_t ref_value)

Set a Widget's state(s) if an integer Subject's value is not equal to a reference value, clear flag otherwise

Parametri

- **obj** -- pointer to Widget
- **subject** -- pointer to Subject
- **state** -- state(s) (can be bit-wise OR-ed) to set or clear (e.g. `LV_STATE_CHECKED`)
- **ref_value** -- reference value to compare Subject's value with

Ritorna

pointer to newly-created Observer

lv_observer_t ***lv_obj_bind_state_if_gt**(lv_obj_t *obj, *lv_subject_t* *subject, *lv_state_t* state, int32_t ref_value)

Set Widget's state(s) if an integer Subject's value is greater than a reference value, clear flag otherwise.

Parametri

- **obj** -- pointer to Widget
- **subject** -- pointer to Subject
- **state** -- state(s) (can be bit-wise OR-ed) to set or clear (e.g. `LV_STATE_CHECKED`)
- **ref_value** -- reference value to compare Subject's value with

Ritorna

pointer to newly-created Observer

lv_observer_t ***lv_obj_bind_state_if_ge**(lv_obj_t *obj, *lv_subject_t* *subject, *lv_state_t* state, int32_t ref_value)

Set Widget's state(s) if an integer Subject's value is greater than or equal to a reference value, clear flag otherwise.

Parametri

- **obj** -- pointer to Widget
- **subject** -- pointer to Subject
- **state** -- state(s) (can be bit-wise OR-ed) to set or clear (e.g. `LV_STATE_CHECKED`)
- **ref_value** -- reference value to compare Subject's value with

Ritorna

pointer to newly-created Observer

lv_observer_t ***lv_obj_bind_state_if_lt**(lv_obj_t *obj, *lv_subject_t* *subject, *lv_state_t* state, int32_t ref_value)

Set Widget's state(s) if an integer Subject's value is less than a reference value, clear flag otherwise.

Parametri

- **obj** -- pointer to Widget
- **subject** -- pointer to Subject
- **state** -- state(s) (can be bit-wise OR-ed) to set or clear (e.g. `LV_STATE_CHECKED`)
- **ref_value** -- reference value to compare Subject's value with

Ritorna

pointer to newly-created Observer

lv_observer_t ***lv_obj_bind_state_if_le**(lv_obj_t *obj, *lv_subject_t* *subject, *lv_state_t* state, int32_t ref_value)

Set Widget's state(s) if an integer Subject's value is less than or equal to a reference value, clear flag otherwise.

Parametri

- **obj** -- pointer to Widget
- **subject** -- pointer to Subject
- **state** -- state(s) (can be bit-wise OR-ed) to set or clear (e.g. `LV_STATE_CHECKED`)
- **ref_value** -- reference value to compare Subject's value with

Ritorna

pointer to newly-created Observer

lv_observer_t ***lv_obj_bind_checked**(lv_obj_t *obj, *lv_subject_t* *subject)

Set an integer Subject to 1 when a Widget is checked and set it 0 when unchecked, and clear Widget's checked state when Subject's value changes to 0 and set it when non-zero.

> **Nota**
>
> Ensure Widget's `LV_OBJ_FLAG_CHECKABLE` flag is set.

Parametri
- **obj** -- pointer to Widget
- **subject** -- pointer to a Subject

Ritorna
pointer to newly-created Observer

lv_observer_t ***lv_label_bind_text**(lv_obj_t *obj, *lv_subject_t* *subject, const char *fmt)

Bind an integer, string, or pointer Subject to a Label.

> **Nota**
>
> `fmt == NULL` can be used only with string and pointer Subjects.

> **Nota**
>
> If Subject is a pointer and `fmt == NULL`, pointer must point to a `\0` terminated string.

Parametri
- **obj** -- pointer to Label
- **subject** -- pointer to Subject
- **fmt** -- optional printf-like format string with 1 format specifier (e.g. "%d °C") or NULL to bind to the value directly.

Ritorna
pointer to newly-created Observer

lv_observer_t ***lv_arc_bind_value**(lv_obj_t *obj, *lv_subject_t* *subject)

Bind an integer subject to an Arc's value.

Parametri
- **obj** -- pointer to Arc
- **subject** -- pointer to Subject

Ritorna
pointer to newly-created Observer

lv_observer_t ***lv_slider_bind_value**(lv_obj_t *obj, *lv_subject_t* *subject)

Bind an integer Subject to a Slider's value.

Parametri
- **obj** -- pointer to Slider
- **subject** -- pointer to Subject

Ritorna
pointer to newly-created Observer

lv_observer_t ***lv_roller_bind_value**(lv_obj_t *obj, *lv_subject_t* *subject)

Bind an integer Subject to a Roller's value.

Parametri
- **obj** -- pointer to Roller
- **subject** -- pointer to Subject

Ritorna
pointer to newly-created Observer

lv_observer_t ***lv_dropdown_bind_value**(lv_obj_t *obj, *lv_subject_t* *subject)

Bind an integer Subject to a Dropdown's value.

Parametri

- **obj** -- pointer to Dropdown
- **subject** -- pointer to Subject

Ritorna

pointer to newly-created Observer

union **lv_subject_value_t**

#include <lv_observer.h> A common type to handle all the various observable types in the same way

Public Members

int32_t **num**

Integer number (opacity, enums, booleans or "normal" numbers)

const void ***pointer**

Constant pointer (string buffer, format string, font, cone text, etc.)

lv_color_t **color**

Color

float **float_v**

Floating point value

struct **lv_subject_t**

#include <lv_observer.h> The Subject (an observable value)

Public Members

lv_ll_t **subs_ll**

Subscribers

lv_subject_value_t **value**

Current value

lv_subject_value_t **prev_value**

Previous value

void ***user_data**

Additional parameter, can be used freely by user

uint32_t **type**

One of the LV_SUBJECT_TYPE_... values

uint32_t **size**

String buffer size or group length

uint32_t **notify_restart_query**

If an Observer was deleted during notification, start notifying from the beginning.

lv_observer_private.h

struct **_lv_observer_t**

#include <lv_observer_private.h> The observer object: a descriptor returned when subscribing LVGL widgets to subjects

Public Members

lv_subject_t ***subject**

Observed subject

lv_observer_cb_t **cb**

Callback that notifies when value changes

void ***target**

A target for the observer, e.g. a widget or any pointer

void ***user_data**

Additional parameter supplied when subscribing

uint32_t **auto_free_user_data**

Automatically free user data when observer is removed

uint32_t **notified**

Was observer already notified?

uint32_t **for_obj**

Is `target` a pointer to a Widget (`lv_obj_t *`)?

14.20.9 snapshot

lv_snapshot.h

Functions

lv_draw_buf_t ***lv_snapshot_take**(lv_obj_t *obj, *lv_color_format_t* cf)

Take snapshot for object with its children, create the draw buffer as needed.

Parametri

- **obj** -- the object to generate snapshot.
- **cf** -- color format for generated image.

Ritorna

a pointer to an draw buffer containing snapshot image, or NULL if failed.

lv_draw_buf_t ***lv_snapshot_create_draw_buf**(lv_obj_t *obj, *lv_color_format_t* cf)

Create a draw buffer to store the snapshot image for object.

Parametri

- **obj** -- the object to generate snapshot.
- **cf** -- color format for generated image.

Ritorna

a pointer to an draw buffer ready for taking snapshot, or NULL if failed.

lv_result_t **lv_snapshot_reshape_draw_buf**(lv_obj_t *obj, lv_draw_buf_t *draw_buf)

Reshape the draw buffer to prepare for taking snapshot for obj. This is usually used to check if the existing draw buffer is enough for obj snapshot. If return LV_RESULT_INVALID, you should create a new one.

Parametri

- **draw_buf** -- the draw buffer to reshape.
- **obj** -- the object to generate snapshot.

lv_result_t **lv_snapshot_take_to_draw_buf**(lv_obj_t *obj, *lv_color_format_t* cf, lv_draw_buf_t *draw_buf)

Take snapshot for object with its children, save image info to provided buffer.

Parametri

- **obj** -- the object to generate snapshot.
- **cf** -- color format for new snapshot image. It could differ with cf of `draw_buf` as long as the new cf will fit in.
- **draw_buf** -- the draw buffer to store the image result. It's reshaped automatically.

Ritorna

LV_RESULT_OK on success, LV_RESULT_INVALID on error.

void **lv_snapshot_free**(*lv_image_dsc_t* *dsc)

Deprecated:

Use `lv_draw_buf_destroy` instead.

Free the snapshot image returned by *lv_snapshot_take*

Parametri

dsc -- the image descriptor generated by lv_snapshot_take.

lv_result_t **lv_snapshot_take_to_buf**(lv_obj_t *obj, *lv_color_format_t* cf, *lv_image_dsc_t* *dsc, void *buf, uint32_t buf_size)

Take snapshot for object with its children, save image info to provided buffer.

Deprecated:

Use lv_snapshot_take_to_draw_buf instead.

Parametri

- **obj** -- the object to generate snapshot.
- **cf** -- color format for generated image.
- **dsc** -- image descriptor to store the image result.
- **buf** -- the buffer to store image data. It must meet align requirement.
- **buf_size** -- provided buffer size in bytes.

Ritorna

LV_RESULT_OK on success, LV_RESULT_INVALID on error.

14.20.10 sysmon

lv_sysmon.h

Functions

lv_obj_t ***lv_sysmon_create**(lv_display_t *disp)

Create a new system monitor label

Parametri

disp -- create the sys. mon. on this display's system layer

Ritorna

the create label

void **lv_sysmon_show_performance**(lv_display_t *disp)

Show system performance monitor: CPU usage and FPS count

Parametri

disp -- target display, NULL: use the default displays

void **lv_sysmon_hide_performance**(lv_display_t *disp)

Hide system performance monitor

Parametri

disp -- target display, NULL: use the default

void **lv_sysmon_performance_dump**(lv_display_t *disp)

Dump the FPS data recorded between the last and current dump call.

Parametri

disp -- target display, NULL: use the default

void **lv_sysmon_performance_resume**(lv_display_t *disp)

Resume the system performance monitor.

Parametri

disp -- target display, NULL: use the default

void **lv_sysmon_performance_pause**(lv_display_t *disp)

Pause the system performance monitor.

> **Nota**
>
> When the sysmon is stopped you can use `lv_sysmon_dump_performance` to get performance information. See `lv_sysmon_dump_performance` for more information.

Parametri

disp -- target display, NULL: use the default

void **lv_sysmon_show_memory**(lv_display_t *disp)

Show system memory monitor: used memory and the memory fragmentation

Parametri

disp -- target display, NULL: use the default displays

void **lv_sysmon_hide_memory**(lv_display_t *disp)

Hide system memory monitor

Parametri

disp -- target display, NULL: use the default displays

lv_sysmon_private.h

Functions

void **lv_sysmon_builtin_init**(void)

Initialize built-in system monitor, such as performance and memory monitor.

void **lv_sysmon_builtin_deinit**(void)

DeInitialize built-in system monitor, such as performance and memory monitor.

struct **_lv_sysmon_backend_data_t**

Public Members

lv_subject_t **subject**

lv_timer_t ***timer**

struct **_lv_sysmon_perf_info_t**

Public Members

bool **inited**

uint32_t **refr_start**

uint32_t **refr_interval_sum**

uint32_t **refr_elaps_sum**

uint32_t **refr_cnt**

uint32_t **render_start**

uint32_t **render_elaps_sum**

uint32_t **render_cnt**

uint32_t **flush_in_render_start**

uint32_t **flush_in_render_elaps_sum**

uint32_t **flush_not_in_render_start**

uint32_t **flush_not_in_render_elaps_sum**

uint32_t **last_report_timestamp**

uint32_t **render_in_progress**

struct *_lv_sysmon_perf_info_t* **measured**

uint32_t **fps**

uint32_t **cpu**

uint32_t **refr_avg_time**

uint32_t **render_avg_time**

Pure rendering time without flush time

uint32_t **flush_avg_time**

Pure flushing time without rendering time

uint32_t **cpu_avg_total**

uint32_t **fps_avg_total**

uint32_t **run_cnt**

struct *_lv_sysmon_perf_info_t* **calculated**

14.20.11 test

lv_test.h

lv_test_display.h

Functions

lv_display_t ***lv_test_display_create**(int32_t hor_res, int32_t ver_res)

lv_test_helpers.h

Defines

LV_HEAP_CHECK(x)

LV_TEST_WIDTH_TO_STRIDE(w, px_size)

Functions

void **lv_test_wait**(uint32_t ms)

Emulate a delay. It's not real delay, but it tricks LVGL to think that the required time has been elapsed. `lv_timer_handler` is called after each millisecond, meaning all the events will be fired inside this function. At the end the animations and display will be also updated.

Parametri

ms -- the number of milliseconds to pass

void **lv_test_fast_forward**(uint32_t ms)

Emulates some time passing. Update the animations and the display only once at the end.

Parametri

ms -- the number of milliseconds to pass

static inline size_t **lv_test_get_free_mem**(void)

lv_test_indev.h

Functions

void **lv_test_indev_create_all**(void)

Create a mouse (pointer), keypad, and encoder indevs. They can be controlled via function calls during the test

void **lv_test_indev_delete_all**(void)

Delete all test input devices

lv_indev_t ***lv_test_indev_get_indev**(*lv_indev_type_t* type)

Get one of the indev created in `lv_test_indev_create_all`

Parametri

type -- type of the indev to get

Ritorna

the indev

void **lv_test_mouse_move_to**(int32_t x, int32_t y)

Move the mouse to the given coordinates. This function doesn't wait, but just changes the state and returns immediately.

Parametri

- **x** -- the target absolute X coordinate
- **y** -- the target absolute Y coordinate

void **lv_test_mouse_move_to_obj**(lv_obj_t *obj)

Move the mouse to the center of a widget This function doesn't wait, but just changes the state and returns immediately.

Parametri

obj -- pointer to an widget

void **lv_test_mouse_move_by**(int32_t x, int32_t y)

Move the mouse cursor. Keep the pressed or released state This function doesn't wait, but just changes the state and returns immediately.

Parametri

- **x** -- the difference in X to move
- **y** -- the difference in Y to move

void **lv_test_mouse_press**(void)

Make the mouse button pressed. This function doesn't wait, but just changes the state and returns immediately.

void **lv_test_mouse_release**(void)

Make the mouse button released. This function doesn't wait, but just changes the state and returns immediately.

void **lv_test_mouse_click_at**(int32_t x, int32_t y)

Emulate a click on a given point. First set the released state, wait a little, press, wait, and release again. The wait time is 50ms. Internally `lv_timer_handler` is called, meaning all the events will be fired inside this function.

Parametri

- **x** -- the target absolute X coordinate
- **y** -- the target absolute Y coordinate

void **lv_test_key_press**(uint32_t k)

Emulate a key press. This function doesn't wait, but just changes the state and returns immediately.

Parametri

k -- the key to press

void **lv_test_key_release**(void)

Release the previously press key. This function doesn't wait, but just changes the state and returns immediately.

Parametri

k -- the key to press

void **lv_test_key_hit**(uint32_t k)

Emulate a key hit. First set the released state, wait a little, press, wait, and release again. The wait time is 50ms. Internally `lv_timer_handler` is called, meaning all the events will be fired inside this function.

Parametri

k -- the key to hit

void **lv_test_encoder_add_diff**(int32_t d)

Emulate encoder rotation, use positive parameter to rotate to the right and negative to rotate to the left. This function doesn't wait, but just changes the state and returns immediately.

Parametri

d -- number of encoder ticks to emulate

void **lv_test_encoder_turn**(int32_t d)

Emulate an encoder turn a wait 50ms. Use positive parameter to rotate to the right and negative to rotate to the left. Internally `lv_timer_handler` is called, meaning all the events will be fired inside this function.

Parametri

d -- number of encoder ticks to emulate

void **lv_test_encoder_press**(void)

Emulate an encoder press. This function doesn't wait, but just changes the state and returns immediately.

void **lv_test_encoder_release**(void)

Emulate an encoder release. This function doesn't wait, but just changes the state and returns immediately.

void **lv_test_encoder_click**(void)

Emulate am encoder click. First set the released state, wait a little, press, wait, and release again. The wait time is 50ms. Internally `lv_timer_handler` is called, meaning all the events will be fired inside this function.

lv_test_indev_gesture.h

Functions

void **lv_test_indev_gesture_create**(void)

Create a touch (pointer) indevs. They can be controlled via function calls during the test

lv_indev_t ***lv_test_indev_get_gesture_indev**(*lv_indev_type_t* type)

Get one of the indev created in `lv_test_indev_gesture_create`

Parametri

type -- type of the indev to get

Ritorna

the indev

void **lv_test_gesture_set_pinch_data**(*lv_point_t* point_0, *lv_point_t* point_1)

Set two touch points data for pinch gesture

Parametri

- **point_0** -- First touch point coordinates
- **point_1** -- Second touch point coordinates

void **lv_test_gesture_pinch_press**(void)

Trigger press state of pinch gesture (both touch points pressed)

void **lv_test_gesture_pinch_release**(void)

Trigger release state of pinch gesture (both touch points released)

void **lv_test_gesture_pinch**(*lv_point_t* point_begin_0, *lv_point_t* point_begin_1, *lv_point_t* point_end_0, *lv_point_t* point_end_1)

Simulate a complete pinch gesture operation

Parametri

- **point_begin_0** -- Starting coordinates of first touch point
- **point_begin_1** -- Starting coordinates of second touch point
- **point_end_0** -- Ending coordinates of first touch point
- **point_end_1** -- Ending coordinates of second touch point

lv_test_private.h

struct **lv_test_state_t**

Public Members

lv_indev_t ***mouse_indev**

lv_indev_t ***keypad_indev**

lv_indev_t ***encoder_indev**

lv_draw_buf_t **draw_buf**

int32_t **x_act**

int32_t **y_act**

uint32_t **key_act**

int32_t **diff_act**

bool **mouse_pressed**

bool **key_pressed**

bool **enc_pressed**

lv_indev_t ***gesture_indev**

lv_indev_touch_data_t ***touch_data**

uint8_t **max_touch_cnt**

lv_test_screenshot_compare.h

Functions

bool **lv_test_screenshot_compare**(const char *fn_ref)

Compare the current content of the test screen with a reference PNG image

- If the reference image is not found it will be created automatically from the rendered screen.
- If the compare fails an `<image_name>_err.png` file will be created with the rendered content next to the reference image.

It requires libPNG.

> **Nota**
>
> This function assumes that the default display is the test display that was created by `lv_test_display_create()`

Parametri

fn_ref -- path to the reference image. Will be appended toREF_IMGS_PATH if set.

Ritorna

true: the reference image and the display are the same; false: they are different (`<image_name>_err.png` is created).

14.20.12 translation

lv_translation.h

Functions

void **lv_translation_init**(void)

Initialize the translation module

void **lv_translation_deinit**(void)

De-initialize the translation module and free all allocated translations

lv_translation_pack_t ***lv_translation_add_static**(const char *languages[], const char *tags[], const char *translations[])

Register a translation pack from static arrays. All the pointers need to be static, that is to live while they are used

Parametri

- **languages** -- List of languages. E.g. `{"en", "de", NULL}`
- **tags** -- Tags that are using in the UI. E.g. `{"dog", "cat", NULL}`

- **translations** -- List of translations. E.g. `{"Dog", "Cat", "Hund", "Katze"}`

Ritorna
The created pack

lv_translation_pack_t *`lv_translation_add_dynamic`(void)

Add a pack to which translations can be added dynamically. `pack->languages` needs to be a malloc-ed array where each language is also malloc-ed as an element. `pack->translation_array` stores the translation having `lv_translation_tag_dsc_t` items In each array element `tag` is a malloced string, `translations` is a malloc-ed array with malloc-ed array for each element.

Ritorna
the created pack to which data can be added manually.

void `lv_translation_set_language`(const char *lang)

Select the current language

Parametri
lang -- a string from the defined languages. E.g. "en" or "de"

const char *`lv_translation_get`(const char *tag)

Get the translated version of a tag on the selected language

> **Nota**
>
> fallback rules:
>
> - if the tag is found on the selected language return it
> - if the tag is not found on the selected language, use the fist language
> - if the tag is not found on the first language, return the tag

Parametri
tag -- the tag to translate

Ritorna
the translation

static inline const char *`lv_tr`(const char *tag)

Shorthand of lv_translation_set_language

Parametri
tag -- the tag to translate

Ritorna
the translation

lv_result_t `lv_translation_add_language`(lv_translation_pack_t *pack, const char *lang)

Add a new language to a dynamic language pack. All languages should be added before adding tags

Parametri
- **pack** -- pointer to a dynamic translation pack
- **lang** -- language to add, e.g. "en", or "de"

Ritorna
LV_RESULT_OK: success, LV_RESULT_INVALID: failed

int32_t `lv_translation_get_language_index`(lv_translation_pack_t *pack, const char *lang_name)

Get the index of a language in a pack.

Parametri
- **pack** -- pointer to a static or dynamic language pack
- **lang_name** -- name of the language to find

Ritorna
index of the language or -1 if not found.

lv_translation_tag_dsc_t *`lv_translation_add_tag`(lv_translation_pack_t *pack, const char *tag_name)

Add a new tag to a dynamic language pack. Once the tag is added the translations for each language can be added too by using `lv_translation_set_tag_translation`

Parametri

- **pack** -- pointer to a dynamic translation pack
- **tag_name** -- name of the tag, e.g. "dog", or "house"

Ritorna

pointer to the allocated tag descriptor

lv_result_t `lv_translation_set_tag_translation`(lv_translation_pack_t *pack, lv_translation_tag_dsc_t *tag, uint32_t lang_idx, const char *trans)

Add a translation to a tag in a dynamic translation pack

Parametri

- **pack** -- pointer to a dynamic translation pack
- **tag** -- return value of `lv_translation_add_tag`
- **lang_idx** -- index of the language for which translation should be set
- **trans** -- the translation on the given language

Ritorna

LV_RESULT_OK: success, LV_RESULT_INVALID: failed

lv_translation_private.h

struct **`_lv_translation_tag_dsc_t`**

Public Members

const char ***tag**

const char ****translations**

Translations for each language

struct **`_lv_translation_pack_t`**

Public Members

const char ****languages**

uint32_t **language_cnt**

uint32_t **is_static**

const char ****tag_p**

const char ****translation_p**

lv_array_t **translation_array**

14.20.13 xml

lv_xml.h

Defines

LV_XML_MAX_PATH_LENGTH

Functions

void **lv_xml_init**(void)

void **lv_xml_deinit**(void)

void ***lv_xml_create**(lv_obj_t *parent, const char *name, const char **attrs)

void ***lv_xml_create_in_scope**(lv_obj_t *parent, lv_xml_component_scope_t *parent_ctx, lv_xml_component_scope_t *scope, const char **attrs)

void **lv_xml_set_default_asset_path**(const char *path_prefix)

Set a path to prefix the image and font file source paths.

In globals.xml usually the source path is like "images/logo.png". But on the actual device it can be located at e.g. "A:ui/assets/images/logo.png". By setting "A:ui/assets/" the path set in the XML files will be prefixed accordingly.

Parametri

path_prefix -- the path to be used as prefix

lv_result_t **lv_xml_register_font**(lv_xml_component_scope_t *scope, const char *name, const lv_font_t *font)

const lv_font_t ***lv_xml_get_font**(lv_xml_component_scope_t *scope, const char *name)

lv_result_t **lv_xml_register_image**(lv_xml_component_scope_t *scope, const char *name, const void *src)

const void ***lv_xml_get_image**(lv_xml_component_scope_t *scope, const char *name)

lv_result_t **lv_xml_register_subject**(lv_xml_component_scope_t *scope, const char *name, *lv_subject_t* *subject)

Map globally available subject name to an actual subject variable

Parametri

- **name** -- name of the subject
- **subject** -- pointer to a subject

Ritorna

LV_RESULT_OK: success

lv_subject_t ***lv_xml_get_subject**(lv_xml_component_scope_t *scope, const char *name)

Get a subject by name.

Parametri

- **scope** -- If specified start searching in that component's subject list, and if not found search in the global space. If NULL search in global space immediately.
- **name** -- Name of the subject to find.

Ritorna

Pointer to the subject or NULL if not found.

lv_result_t **lv_xml_register_const**(lv_xml_component_scope_t *scope, const char *name, const char *value)

const char ***lv_xml_get_const**(lv_xml_component_scope_t *scope, const char *name)

lv_result_t **lv_xml_register_event_cb**(lv_xml_component_scope_t *scope, const char *name, *lv_event_cb_t* cb)

lv_event_cb_t **lv_xml_get_event_cb**(lv_xml_component_scope_t *scope, const char *name)

lv_xml_base_types.h

Functions

lv_state_t **lv_xml_state_to_enum**(const char *txt)

Convert an state string to enum

Parametri

txt -- e.g. "pressed"

Ritorna

the related enum, e.g. `LV_STATE_PRESSED`

int32_t **lv_xml_to_size**(const char *txt)

Process inputs "content", "32", "32px", or "25%" and convert them to integer

Parametri

txt -- the input string

Ritorna

the integer size

lv_align_t **lv_xml_align_to_enum**(const char *txt)

Convert an align string to enum

Parametri

txt -- e.g. "center"

Ritorna

the related enum, e.g. `LV_ALIGN_CENTER`

lv_dir_t **lv_xml_dir_to_enum**(const char *txt)

Convert a direction string to enum

Parametri

txt -- e.g. "top"

Ritorna

the related enum, e.g. `LV_DIR_TOP`

lv_border_side_t **lv_xml_border_side_to_enum**(const char *txt)

Convert a direction string to enum

Parametri

txt -- e.g. "top"

Ritorna

the related enum, e.g. `LV_BORDER_SIDE_TOP`

lv_base_dir_t **lv_xml_base_dir_to_enum**(const char *txt)

Convert a base dir string to enum

Parametri

txt -- e.g. "rtl"

Ritorna

the related enum, e.g. `LV_BASE_DIR_RTL`

lv_grad_dir_t **lv_xml_grad_dir_to_enum**(const char *txt)

Convert a grad dir string to enum

Parametri

txt -- e.g. "hor"

Ritorna

the related enum, e.g. `LV_GRAD_DIR_HOR`

lv_text_align_t **lv_xml_text_align_to_enum**(const char *txt)

Convert a text align string to enum

Parametri
txt -- e.g. "left"
Ritorna
the related enum, e.g. `LV_TEXT_ALIGN_LEFT`

lv_text_decor_t **lv_xml_text_decor_to_enum**(const char *txt)

Convert a text decor string to enum
Parametri
txt -- e.g. "underline"
Ritorna
the related enum, e.g. `LV_TEXT_DECOR_UNDERLINE`

lv_scroll_snap_t **lv_xml_scroll_snap_to_enum**(const char *txt)

Convert a scroll snap string to enum
Parametri
txt -- e.g. "start"
Ritorna
the related enum, e.g. `LV_SCROLL_SNAP_START`

lv_flex_flow_t **lv_xml_flex_flow_to_enum**(const char *txt)

Convert a flex flow string to enum
Parametri
txt -- e.g. "row_wrap"
Ritorna
the related enum, e.g. `LV_FLEX_FLOW_ROW_WRAP`

lv_flex_align_t **lv_xml_flex_align_to_enum**(const char *txt)

Convert a flex align string to enum
Parametri
txt -- e.g. "space_between"
Ritorna
the related enum, e.g. `LV_FLEX_ALIGN_SPACE_BETWEEN`

lv_grid_align_t **lv_xml_grid_align_to_enum**(const char *txt)

Convert a grid align string to enum
Parametri
txt -- e.g. "space_between"
Ritorna
the related enum, e.g. `LV_GRID_ALIGN_SPACE_BETWEEN`

lv_layout_t **lv_xml_layout_to_enum**(const char *txt)

Convert a layout string to enum
Parametri
txt -- e.g. "flex"
Ritorna
the related enum, e.g. `LV_LAYOUT_FLEX`

lv_blend_mode_t **lv_xml_blend_mode_to_enum**(const char *txt)

Convert a blend mode string to enum
Parametri
txt -- e.g. "additive"
Ritorna
the related enum, e.g. `LV_BLEND_MODE_ADDITIVE`

lv_event_code_t **lv_xml_trigger_text_to_enum_value**(const char *txt)

Convert an event trigger string to enum
Parametri
txt -- e.g. "clicked"
Ritorna
the related enum, e.g. `LV_EVENT_CLICKED`

lv_screen_load_anim_t **lv_xml_screen_load_anim_text_to_enum_value**(const char *txt)

Convert a screen animation type string to enum

Parametri

txt -- e.g. "over_right"

Ritorna

the related enum, e.g. `LV_SCREEN_LOAD_ANIM_OVER_RIGHT`

lv_xml_component.h

Functions

lv_obj_t ***lv_xml_component_process**(lv_xml_parser_state_t *state, const char *name, const char **attrs)

Process a component during parsing an XML. It create a widget and apply all the attributes

Parametri

- **state** -- the current parsing state
- **name** -- name of the component
- **attrs** -- attributes of the widget

Ritorna

lv_result_t **lv_xml_component_register_from_data**(const char *name, const char *xml_def)

Load the styles, constants, another data of the component. It needs to be called only once for each component.

Parametri

- **name** -- the name as the component will be referenced later in other components
- **xml_def** -- the XML definition of the component as a NULL terminated string

Ritorna

LV_RES_OK: loaded successfully, LV_RES_INVALID: otherwise

lv_result_t **lv_xml_component_register_from_file**(const char *path)

Load the styles, constants, another data of the component. It needs to be called only once for each component.

Parametri

path -- path to an XML file

Ritorna

LV_RES_OK: loaded successfully, LV_RES_INVALID: otherwise

lv_xml_component_scope_t ***lv_xml_component_get_scope**(const char *component_name)

Get the scope of a component which was registered by `lv_xml_component_register_from_data` or `lv_xml_component_register_from_file`

Parametri

component_name -- name of the component

Ritorna

pointer the scope or NULL if not found

lv_result_t **lv_xml_component_unregister**(const char *name)

Remove a component from from the list.

Parametri

name -- the name of the component (used during registration)

Ritorna

LV_RESULT_OK on successful unregistration, LV_RESULT_INVALID otherwise.

lv_xml_component_private.h

Typedef

typedef void *(***lv_xml_component_process_cb_t**)(lv_obj_t *parent, const char *data, const char **attrs)

Functions

void **lv_xml_component_init**(void)

Initialize the components system.

void **lv_xml_component_scope_init**(lv_xml_component_scope_t *scope)

Initialize the linked lists of a component context

Parametri

scope -- pointer to a component contexts

struct **_lv_xml_component_scope_t**

Public Members

const char ***name**

lv_ll_t **style_ll**

lv_ll_t **const_ll**

lv_ll_t **param_ll**

lv_ll_t **gradient_ll**

lv_ll_t **subjects_ll**

lv_ll_t **font_ll**

lv_ll_t **image_ll**

lv_ll_t **event_ll**

const char ***view_def**

const char ***extends**

uint32_t **is_widget**

uint32_t **is_screen**

struct *_lv_xml_component_scope_t* ***next**

struct **lv_xml_const_t**

Public Members

const char ***name**

const char ***value**

struct **lv_xml_subject_t**

Public Members

const char ***name**

lv_subject_t ***subject**

struct **lv_xml_param_t**

Public Members

const char ***name**

const char ***def**

const char ***type**

struct **lv_xml_grad_t**

Public Members

const char ***name**

lv_grad_dsc_t **grad_dsc**

lv_xml_parser.h

Enums

enum **lv_xml_parser_section_t**

Values:

enumerator **LV_XML_PARSER_SECTION_NONE**

enumerator **LV_XML_PARSER_SECTION_API**

enumerator **LV_XML_PARSER_SECTION_CONSTS**

enumerator **LV_XML_PARSER_SECTION_GRAD**

enumerator **LV_XML_PARSER_SECTION_GRAD_STOP**

enumerator **LV_XML_PARSER_SECTION_STYLES**

enumerator **LV_XML_PARSER_SECTION_FONTS**

enumerator **LV_XML_PARSER_SECTION_IMAGES**

enumerator **LV_XML_PARSER_SECTION_SUBJECTS**

enumerator **LV_XML_PARSER_SECTION_VIEW**

Functions

void **lv_xml_parser_state_init**(lv_xml_parser_state_t *state)

void **lv_xml_parser_start_section**(lv_xml_parser_state_t *state, const char *name)

void **lv_xml_parser_end_section**(lv_xml_parser_state_t *state, const char *name)

void ***lv_xml_state_get_parent**(lv_xml_parser_state_t *state)

void ***lv_xml_state_get_item**(lv_xml_parser_state_t *state)

struct **_lv_xml_parser_state_t**

Public Members

const char ***tag_name**

lv_xml_component_scope_t **scope**

lv_ll_t **parent_ll**

lv_obj_t ***parent**

lv_obj_t ***item**

lv_obj_t ***view**

const char ****parent_attrs**

lv_xml_component_scope_t ***parent_scope**

lv_xml_parser_section_t **section**

lv_xml_private.h

struct **lv_xml_font_t**

Public Members

const char *`name`

const lv_font_t *`font`

void (*`font_destroy_cb`)(lv_font_t*)

struct `lv_xml_image_t`

Public Members

const char *`name`

const void *`src`

struct `lv_xml_event_cb_t`

Public Members

const char *`name`

lv_event_cb_t `cb`

lv_xml_style.h

Typedef

typedef struct *_lv_xml_style_t* `lv_xml_style_t`

Functions

lv_result_t `lv_xml_style_register`(lv_xml_component_scope_t *scope, const char **attrs)

Add a style to `ctx` and set the style properties from `attrs`

Parametri

- `scope` -- add styles here. (Constants should be already added as style properties might use them)
- `attrs` -- list of attribute names and values

lv_state_t `lv_xml_style_state_to_enum`(const char *txt)

Convert a style state to enum

Parametri

`txt` -- e.g. "pressed"

Ritorna

the enum `LV_STATE_PRESSED`

lv_part_t `lv_xml_style_part_to_enum`(const char *txt)

Convert a style part to enum

Parametri

`txt` -- e.g. "knob"

Ritorna

the enum `LV_PART_KNOB`

const char *`lv_xml_style_string_process`(char *txt, *lv_style_selector_t* *selector)

Decompose a string like `"style1:pressed:checked:knob"` to style name and selector

Parametri

- **`txt`** -- the input string
- **`selector`** -- store the selectors here

Ritorna

the style name or `NULL` on any error

lv_xml_style_t *`lv_xml_get_style_by_name`(lv_xml_component_scope_t *scope, const char *name)

Find a style by name which was added by `lv_xml_style_register`

Parametri

- **`scope`** -- the default context to search in
- **`name`** -- the name of the style. Can start with a component name prefix (e.g. `my_button.blue`) to overwrite the ctx

Ritorna

the style structure

lv_grad_dsc_t *`lv_xml_component_get_grad`(lv_xml_component_scope_t *scope, const char *name)

Get a gradient descriptor defined for a component

Parametri

- **`scope`** -- component context where the gradient should be found
- **`name`** -- name of the gradient

Ritorna

a gradient descriptor

struct **`_lv_xml_style_t`**

Public Members

const char ***`name`**

const char ***`long_name`**

lv_style_t **`style`**

lv_xml_test.h

Enums

enum **`lv_xml_test_step_type_t`**

Values:

enumerator **`LV_XML_TEST_STEP_TYPE_NONE`**

enumerator **`LV_XML_TEST_STEP_TYPE_MOVE_TO`**

enumerator **`LV_XML_TEST_STEP_TYPE_PRESS`**

enumerator **`LV_XML_TEST_STEP_TYPE_RELEASE`**

enumerator **`LV_XML_TEST_STEP_TYPE_CLICK_AT`**

enumerator **LV_XML_TEST_STEP_TYPE_WAIT**

enumerator **LV_XML_TEST_STEP_TYPE_FREEZE**

enumerator **LV_XML_TEST_STEP_TYPE_SCREENSHOT_COMPARE**

enumerator **LV_XML_TEST_STEP_TYPE_SUBJECT_SET**

enumerator **LV_XML_TEST_STEP_TYPE_SUBJECT_COMPARE**

Functions

lv_result_t **lv_xml_test_register_from_data**(const char *xml_def, const char *ref_image_path_prefix)

Load the styles, constants, another data of the test. It needs to be called only once for each test.

Parametri

- **xml_def** -- the XML definition of the test as a NULL terminated string
- **ref_image_path_prefix** -- prefix for the path of reference images

Ritorna

LV_RES_OK: loaded successfully, LV_RES_INVALID: otherwise

lv_result_t **lv_xml_test_register_from_file**(const char *path, const char *ref_image_path_prefix)

Load the styles, constants, another data of the test. It needs to be called only once for each test.

Parametri

- **path** -- path to an XML file
- **ref_image_path_prefix** -- prefix for the path of reference images

Ritorna

LV_RES_OK: loaded successfully, LV_RES_INVALID: otherwise

void **lv_xml_test_unregister**(void)

Free resources allocated for testing.

void **lv_xml_test_run_init**(void)

Switch to testing mode. Needs to be called to use *lv_xml_test_run_next()*

bool **lv_xml_test_run_next**(uint32_t slowdown)

Run the next test step.

Parametri

slowdown -- 0: max speed, 1: real speed, 2: half speed, ... ,10: ten times slower

Ritorna

true: the step passed; false: the step failed

void **lv_xml_test_run_stop**(void)

Leave testing mode.

uint32_t **lv_xml_test_run_all**(uint32_t slowdown)

Run all the test steps. It calls *lv_xml_test_run_init()*, *lv_xml_test_run_next()*, and `lv_xml_test_run_stop` internally so no further preparation or cleanup is needed.

Parametri

slowdown -- 0: max speed, 1: real speed, 2: half speed, ... ,10: ten times slower

Ritorna

number of failed tests

uint32_t **lv_xml_test_get_step_count**(void)

Get the number of steps in a test

Ritorna

the number of `<step>`s

lv_xml_test_step_type_t **lv_xml_test_get_step_type**(uint32_t idx)

Get the type of a step

Parametri

idx -- the index of a step (< step_count)

Ritorna

element of `lv_xml_test_step_type_t`

bool **lv_xml_test_get_status**(uint32_t idx)

Check if the a step was passed. Can be called after lv_xml_test_run()

Parametri

idx -- the index of a step (< step_count)

Ritorna

true: the step passed, false: the step failed

lv_xml_translation.h

Functions

lv_result_t **lv_xml_translation_register_from_file**(const char *path)

Register translations from an XML file

Parametri

path -- path to an XML file (staring with a driver letter)

Ritorna

LV_RES_OK: no error

lv_result_t **lv_xml_translation_register_from_data**(const char *xml_def)

Register translations from an XML string

Parametri

xml_def -- the XML definition as a string

Ritorna

LV_RES_OK: no error

lv_xml_update.h

Functions

lv_result_t **lv_xml_update_from_data**(const char *xml_def)

Load the styles, constants, another data of the component. It needs to be called only once for each component.

Parametri

xml_def -- the XML definition of the component as a NULL terminated string

Ritorna

LV_RES_OK: loaded successfully, LV_RES_INVALID: otherwise

lv_xml_utils.h

Functions

const char ***lv_xml_get_value_of**(const char **attrs, const char *name)

int32_t **lv_xml_atoi**(const char *str)

int32_t **lv_xml_atoi_split**(const char **str, char delimiter)

Convert sections of a string to int. The end of the string is indicated by the `delimiter`.

Parametri

- **str** -- pointer to a string, it will point to the character after the delimiter
- **delimiter** -- a character to indicate the end of the int

Ritorna

the int before the next delimiter

float **lv_xml_atof**(const char *str)

float **lv_xml_atof_split**(const char **str, char delimiter)

Convert sections of a string to float. The end of the string is indicated by the `delimiter`.

Parametri

- **str** -- pointer to a string, it will point to the character after the delimiter
- **delimiter** -- a character to indicate the end of the float

Ritorna

the float before the next delimiter

lv_color_t **lv_xml_to_color**(const char *str)

lv_opa_t **lv_xml_to_opa**(const char *str)

Concert percentage or integer opacity value from string to integer.

Parametri

str -- e.g. "70%" or 180

Ritorna

0..255

bool **lv_xml_to_bool**(const char *str)

int32_t **lv_xml_strtol**(const char *str, char **endptr, int32_t base)

char ***lv_xml_split_str**(char **src, char delimiter)

Find a delimiter in a string, terminate the string on the delimiter and update the source string to the next part

Parametri

- **src** -- point to a variable containing the input string
- **delimiter** -- a delimiter character, e.g. ':'

Ritorna

the beginning of next section in the string closed at the delimiter

lv_xml_widget.h

Typedef

typedef void *(***lv_xml_widget_create_cb_t**)(lv_xml_parser_state_t *state, const char **parent_attrs)

typedef void (***lv_xml_widget_apply_cb_t**)(lv_xml_parser_state_t *state, const char **parent_attrs)

typedef struct *_lv_widget_processor_t* **lv_widget_processor_t**

Functions

lv_result_t **lv_xml_widget_register**(const char *name, *lv_xml_widget_create_cb_t* create_cb, *lv_xml_widget_apply_cb_t* apply_cb)

lv_widget_processor_t ***lv_xml_widget_get_processor**(const char *name)

lv_widget_processor_t ***lv_xml_widget_get_extended_widget_processor**(const char *extends)

struct **_lv_widget_processor_t**

Public Members

const char ***name**

lv_xml_widget_create_cb_t **create_cb**

lv_xml_widget_apply_cb_t **apply_cb**

struct *_lv_widget_processor_t* ***next**

parsers

lv_xml_arc_parser.h

Functions

void ***lv_xml_arc_create**(lv_xml_parser_state_t *state, const char **attrs)

void **lv_xml_arc_apply**(lv_xml_parser_state_t *state, const char **attrs)

lv_xml_bar_parser.h

Functions

void ***lv_xml_bar_create**(lv_xml_parser_state_t *state, const char **attrs)

void **lv_xml_bar_apply**(lv_xml_parser_state_t *state, const char **attrs)

lv_xml_button_parser.h

Functions

void ***lv_xml_button_create**(lv_xml_parser_state_t *state, const char **attrs)

void **lv_xml_button_apply**(lv_xml_parser_state_t *state, const char **attrs)

lv_xml_buttonmatrix_parser.h

Functions

void ***lv_xml_buttonmatrix_create**(lv_xml_parser_state_t *state, const char **attrs)

void **lv_xml_buttonmatrix_apply**(lv_xml_parser_state_t *state, const char **attrs)

lv_xml_calendar_parser.h

Functions

void ***lv_xml_calendar_create**(lv_xml_parser_state_t *state, const char **attrs)

void **lv_xml_calendar_apply**(lv_xml_parser_state_t *state, const char **attrs)

void ***lv_xml_calendar_header_arrow_create**(lv_xml_parser_state_t *state, const char **attrs)

void **lv_xml_calendar_header_arrow_apply**(lv_xml_parser_state_t *state, const char **attrs)

void ***lv_xml_calendar_header_dropdown_create**(lv_xml_parser_state_t *state, const char **attrs)

void **lv_xml_calendar_header_dropdown_apply**(lv_xml_parser_state_t *state, const char **attrs)

lv_xml_canvas_parser.h

Functions

void ***lv_xml_canvas_create**(lv_xml_parser_state_t *state, const char **attrs)

void **lv_xml_canvas_apply**(lv_xml_parser_state_t *state, const char **attrs)

lv_xml_chart_parser.h

Functions

void *`lv_xml_chart_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_chart_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_xml_chart_series_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_chart_series_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_xml_chart_cursor_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_chart_cursor_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_xml_chart_axis_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_chart_axis_apply`(lv_xml_parser_state_t *state, const char **attrs)

lv_xml_checkbox_parser.h

Functions

void *`lv_xml_checkbox_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_checkbox_apply`(lv_xml_parser_state_t *state, const char **attrs)

lv_xml_dropdown_parser.h

Functions

void *`lv_xml_dropdown_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_dropdown_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_xml_dropdown_list_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_dropdown_list_apply`(lv_xml_parser_state_t *state, const char **attrs)

lv_xml_image_parser.h

Functions

void *`lv_xml_image_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_image_apply`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_check_file`(const char *filepath)

lv_xml_keyboard_parser.h

Functions

void *`lv_xml_keyboard_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_keyboard_apply`(lv_xml_parser_state_t *state, const char **attrs)

lv_xml_label_parser.h

Functions

void *`lv_xml_label_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_label_apply`(lv_xml_parser_state_t *state, const char **attrs)

lv_xml_obj_parser.h

Functions

void *`lv_xml_obj_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_obj_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_obj_xml_style_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_obj_xml_style_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_obj_xml_remove_style_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_obj_xml_remove_style_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_obj_xml_remove_style_all_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_obj_xml_remove_style_all_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_obj_xml_event_cb_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_obj_xml_event_cb_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_obj_xml_subject_set_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_obj_xml_subject_set_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_obj_xml_subject_increment_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_obj_xml_subject_increment_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_obj_xml_bind_style_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_obj_xml_bind_style_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_obj_xml_bind_flag_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_obj_xml_bind_flag_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_obj_xml_bind_state_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_obj_xml_bind_state_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_obj_xml_screen_load_event_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_obj_xml_screen_load_event_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_obj_xml_screen_create_event_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_obj_xml_screen_create_event_apply`(lv_xml_parser_state_t *state, const char **attrs)

lv_xml_roller_parser.h

Functions

void *`lv_xml_roller_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_roller_apply`(lv_xml_parser_state_t *state, const char **attrs)

lv_xml_scale_parser.h

Functions

void *`lv_xml_scale_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_scale_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_xml_scale_section_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_scale_section_apply`(lv_xml_parser_state_t *state, const char **attrs)

lv_xml_slider_parser.h

Functions

void *`lv_xml_slider_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_slider_apply`(lv_xml_parser_state_t *state, const char **attrs)

lv_xml_spangroup_parser.h

Functions

void *`lv_xml_spangroup_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_spangroup_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_xml_spangroup_span_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_spangroup_span_apply`(lv_xml_parser_state_t *state, const char **attrs)

lv_xml_table_parser.h

Functions

void *`lv_xml_table_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_table_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_xml_table_column_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_table_column_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_xml_table_cell_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_table_cell_apply`(lv_xml_parser_state_t *state, const char **attrs)

lv_xml_tabview_parser.h

Functions

void *`lv_xml_tabview_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_tabview_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_xml_tabview_tab_bar_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_tabview_tab_bar_apply`(lv_xml_parser_state_t *state, const char **attrs)

void *`lv_xml_tabview_tab_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_tabview_tab_apply`(lv_xml_parser_state_t *state, const char **attrs)

lv_xml_textarea_parser.h

Functions

void *`lv_xml_textarea_create`(lv_xml_parser_state_t *state, const char **attrs)

void `lv_xml_textarea_apply`(lv_xml_parser_state_t *state, const char **attrs)

14.21 stdlib

14.21.1 lv_mem.h

Typedef

typedef void *`lv_mem_pool_t`

Functions

void `lv_mem_init`(void)

Initialize to use malloc/free/realloc etc

void `lv_mem_deinit`(void)

Drop all dynamically allocated memory and reset the memory pools' state

lv_mem_pool_t `lv_mem_add_pool`(void *mem, size_t bytes)

void `lv_mem_remove_pool`(*lv_mem_pool_t* pool)

void *`lv_malloc`(size_t size)

Allocate memory dynamically

Parametri

`size` -- requested size in bytes

Ritorna

pointer to allocated uninitialized memory, or NULL on failure

void *`lv_calloc`(size_t num, size_t size)

Allocate a block of zeroed memory dynamically

Parametri

- `num` -- requested number of element to be allocated.
- `size` -- requested size of each element in bytes.

Ritorna

pointer to allocated zeroed memory, or NULL on failure

void *`lv_zalloc`(size_t size)

Allocate zeroed memory dynamically

Parametri

`size` -- requested size in bytes

Ritorna

pointer to allocated zeroed memory, or NULL on failure

void *`lv_malloc_zeroed`(size_t size)

Allocate zeroed memory dynamically

Parametri

`size` -- requested size in bytes

Ritorna

pointer to allocated zeroed memory, or NULL on failure

void `lv_free`(void *data)

Free an allocated data

Parametri
data -- pointer to an allocated memory

void *`lv_realloc`(void *data_p, size_t new_size)

Reallocate a memory with a new size. The old content will be kept.

Parametri
- **data_p** -- pointer to an allocated memory. Its content will be copied to the new memory block and freed
- **new_size** -- the desired new size in byte

Ritorna
pointer to the new memory, NULL on failure

void *`lv_reallocf`(void *data_p, size_t new_size)

Reallocate a memory with a new size. The old content will be kept. In case of failure, the old pointer is free'd.

Parametri
- **data_p** -- pointer to an allocated memory. Its content will be copied to the new memory block and freed
- **new_size** -- the desired new size in byte

Ritorna
pointer to the new memory, NULL on failure

void *`lv_malloc_core`(size_t size)

Used internally to execute a plain `malloc` operation

Parametri
size -- size in bytes to `malloc`

void `lv_free_core`(void *p)

Used internally to execute a plain `free` operation

Parametri
p -- memory address to free

void *`lv_realloc_core`(void *p, size_t new_size)

Used internally to execute a plain realloc operation

Parametri
- **p** -- memory address to realloc
- **new_size** -- size in bytes to realloc

void `lv_mem_monitor_core`(*lv_mem_monitor_t* *mon_p)

Used internally by *lv_mem_monitor()* to gather LVGL heap state information.

Parametri
mon_p -- pointer to *lv_mem_monitor_t* object to be populated.

lv_result_t `lv_mem_test_core`(void)

lv_result_t `lv_mem_test`(void)

Tests the memory allocation system by allocating and freeing a block of memory.

Ritorna
LV_RESULT_OK if the memory allocation system is working properly, or LV_RESULT_INVALID if there is an error.

void `lv_mem_monitor`(*lv_mem_monitor_t* *mon_p)

Give information about the work memory of dynamic allocation

Parametri
mon_p -- pointer to a *lv_mem_monitor_t* variable, the result of the analysis will be stored here

struct `lv_mem_monitor_t`

#include <lv_mem.h> Heap information structure.

Public Members

size_t **total_size**
Total heap size

size_t **free_cnt**

size_t **free_size**
Size of available memory

size_t **free_biggest_size**

size_t **used_cnt**

size_t **max_used**
Max size of Heap memory used

uint8_t **used_pct**
Percentage used

uint8_t **frag_pct**
Amount of fragmentation

14.21.2 lv_mem_private.h

14.21.3 lv_sprintf.h

Defines

LV_PRId32

LV_PRIu32

LV_PRIx32

LV_PRIX32

LV_PRId64

LV_PRIu64

LV_PRIx64

LV_PRIX64

Functions

int **lv_snprintf**(char *buffer, size_t count, const char *format, ...)

int **lv_vsnprintf**(char *buffer, size_t count, const char *format, va_list va)

14.21.4 lv_string.h

Functions

void *`lv_memcpy`(void *dst, const void *src, size_t len)

Copies a block of memory from a source address to a destination address.

> **Nota**
>
> The function does not check for any overlapping of the source and destination memory blocks.

Parametri

- `dst` -- Pointer to the destination array where the content is to be copied.
- `src` -- Pointer to the source of data to be copied.
- `len` -- Number of bytes to copy.

Ritorna

Pointer to the destination array.

void `lv_memset`(void *dst, uint8_t v, size_t len)

Fills a block of memory with a specified value.

Parametri

- `dst` -- Pointer to the destination array to fill with the specified value.
- `v` -- Value to be set. The value is passed as an int, but the function fills the block of memory using the unsigned char conversion of this value.
- `len` -- Number of bytes to be set to the value.

void *`lv_memmove`(void *dst, const void *src, size_t len)

Move a block of memory from source to destination.

Parametri

- `dst` -- Pointer to the destination array where the content is to be copied.
- `src` -- Pointer to the source of data to be copied.
- `len` -- Number of bytes to copy

Ritorna

Pointer to the destination array.

int `lv_memcmp`(const void *p1, const void *p2, size_t len)

This function will compare two memory blocks.

Parametri

- `p1` -- Pointer to the first memory block
- `p2` -- Pointer to the second memory block
- `len` -- Number of bytes to compare

Ritorna

The difference between the value of the first unmatching byte.

static inline void `lv_memzero`(void *dst, size_t len)

Same as `memset(dst, 0x00, len)`.

Parametri

- `dst` -- pointer to the destination buffer
- `len` -- number of byte to set

size_t `lv_strlen`(const char *str)

Computes the length of the string str up to (but not including) the terminating null character.

Parametri

`str` -- Pointer to the null-terminated byte string to be examined.

Ritorna

The length of the string in bytes.

size_t `lv_strnlen`(const char *str, size_t max_len)

Computes the length of the string str up to (but not including) the terminating null character, or the given maximum length.

Parametri

- **str** -- Pointer to byte string that is null-terminated or at least max_len bytes long.
- **max_len** -- Maximum number of characters to examine.

Ritorna

The length of the string in bytes.

size_t **lv_strlcpy**(char *dst, const char *src, size_t dst_size)

Copies up to dst_size-1 (non-null) characters from src to dst. A null terminator is always added.

Parametri

- **dst** -- Pointer to the destination array where the content is to be copied.
- **src** -- Pointer to the source of data to be copied.
- **dst_size** -- Maximum number of characters to be copied to dst, including the null character.

Ritorna

The length of src. The return value is equivalent to the value returned by lv_strlen(src)

char ***lv_strncpy**(char *dst, const char *src, size_t dest_size)

Copies up to dest_size characters from the string pointed to by src to the character array pointed to by dst and fills the remaining length with null bytes.

> **Nota**
>
> dst will not be null terminated if dest_size bytes were copied from src before the end of src was reached.

Parametri

- **dst** -- Pointer to the destination array where the content is to be copied.
- **src** -- Pointer to the source of data to be copied.
- **dest_size** -- Maximum number of characters to be copied to dst.

Ritorna

A pointer to the destination array, which is dst.

char ***lv_strcpy**(char *dst, const char *src)

Copies the string pointed to by src, including the terminating null character, to the character array pointed to by dst.

Parametri

- **dst** -- Pointer to the destination array where the content is to be copied.
- **src** -- Pointer to the source of data to be copied.

Ritorna

A pointer to the destination array, which is dst.

int **lv_strcmp**(const char *s1, const char *s2)

This function will compare two strings without specified length.

Parametri

- **s1** -- pointer to the first string
- **s2** -- pointer to the second string

Ritorna

the difference between the value of the first unmatching character.

int **lv_strncmp**(const char *s1, const char *s2, size_t len)

This function will compare two strings up to the given length.

Parametri

- **s1** -- pointer to the first string
- **s2** -- pointer to the second string
- **len** -- the maximum amount of characters to compare

Ritorna

the difference between the value of the first unmatching character.

static inline bool **lv_streq**(const char *s1, const char *s2)

Returns true if the two strings are equal. Just a wrapper around strcmp for convenience.

Parametri
- **s1** -- pointer to the first string
- **s2** -- pointer to the second string

Ritorna
true: the strings are equal; false: otherwise

char ***lv_strdup**(const char *src)

Duplicate a string by allocating a new one and copying the content.

Parametri
src -- Pointer to the source of data to be copied.

Ritorna
A pointer to the new allocated string. NULL if failed.

char ***lv_strndup**(const char *src, size_t max_len)

Duplicate a string by allocating a new one and copying the content up to the end or the specified maximum length, whichever comes first.

Parametri
- **src** -- Pointer to the source of data to be copied.
- **max_len** -- Maximum number of characters to be copied.

Ritorna
Pointer to a newly allocated null-terminated string. NULL if failed.

char ***lv_strcat**(char *dst, const char *src)

Copies the string pointed to by src, including the terminating null character, to the end of the string pointed to by dst.

Parametri
- **dst** -- Pointer to the destination string where the content is to be appended.
- **src** -- Pointer to the source of data to be copied.

Ritorna
A pointer to the destination string, which is dst.

char ***lv_strncat**(char *dst, const char *src, size_t src_len)

Copies up to src_len characters from the string pointed to by src to the end of the string pointed to by dst. A terminating null character is appended to dst even if no null character was encountered in src after src_len characters were copied.

Parametri
- **dst** -- Pointer to the destination string where the content is to be appended.
- **src** -- Pointer to the source of data to be copied.
- **src_len** -- Maximum number of characters from src to be copied to the end of dst.

Ritorna
A pointer to the destination string, which is dst.

char ***lv_strchr**(const char *str, int c)

Searches for the first occurrence of character c in the string str.

Parametri
- **str** -- Pointer to the null-terminated byte string to be searched.
- **c** -- The character to be searched for.

Ritorna
A pointer to the first occurrence of character c in the string str, or a null pointer if c is not found.

14.21.5 builtin

lv_tlsf.h

Typedef

typedef void ***lv_tlsf_t**

typedef void ***lv_pool_t**

typedef void (***lv_tlsf_walker**)(void *ptr, size_t size, int used, void *user)

Functions

lv_tlsf_t **lv_tlsf_create**(void *mem)

lv_tlsf_t **lv_tlsf_create_with_pool**(void *mem, size_t bytes)

void **lv_tlsf_destroy**(*lv_tlsf_t* tlsf)

lv_pool_t **lv_tlsf_get_pool**(*lv_tlsf_t* tlsf)

lv_pool_t **lv_tlsf_add_pool**(*lv_tlsf_t* tlsf, void *mem, size_t bytes)

void **lv_tlsf_remove_pool**(*lv_tlsf_t* tlsf, *lv_pool_t* pool)

void ***lv_tlsf_malloc**(*lv_tlsf_t* tlsf, size_t bytes)

void ***lv_tlsf_memalign**(*lv_tlsf_t* tlsf, size_t align, size_t bytes)

void ***lv_tlsf_realloc**(*lv_tlsf_t* tlsf, void *ptr, size_t size)

size_t **lv_tlsf_free**(*lv_tlsf_t* tlsf, const void *ptr)

size_t **lv_tlsf_block_size**(void *ptr)

size_t **lv_tlsf_size**(void)

size_t **lv_tlsf_align_size**(void)

size_t **lv_tlsf_block_size_min**(void)

size_t **lv_tlsf_block_size_max**(void)

size_t **lv_tlsf_pool_overhead**(void)

size_t **lv_tlsf_alloc_overhead**(void)

void **lv_tlsf_walk_pool**(*lv_pool_t* pool, *lv_tlsf_walker* walker, void *user)

int **lv_tlsf_check**(*lv_tlsf_t* tlsf)

int **lv_tlsf_check_pool**(*lv_pool_t* pool)

lv_tlsf_private.h

struct **lv_tlsf_state_t**

Public Members

lv_tlsf_t **tlsf**

size_t **cur_used**

size_t **max_used**

lv_ll_t **pool_ll**

14.22 themes

14.22.1 lv_theme.h

Typedef

typedef void (*`lv_theme_apply_cb_t`)(lv_theme_t*, lv_obj_t*)

Functions

lv_theme_t *`lv_theme_get_from_obj`(lv_obj_t *obj)

Get the theme assigned to the display of the object

Parametri

obj -- pointer to a theme object

Ritorna

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

void `lv_theme_apply`(lv_obj_t *obj)

Apply the active theme on an object

Parametri

obj -- pointer to an object

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

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

Parametri

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

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

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

Parametri

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

const lv_font_t *`lv_theme_get_font_small`(lv_obj_t *obj)

Get the small font of the theme

Parametri

obj -- pointer to an object

Ritorna

pointer to the font

const lv_font_t *`lv_theme_get_font_normal`(lv_obj_t *obj)

Get the normal font of the theme

Parametri

obj -- pointer to an object

Ritorna

pointer to the font

const lv_font_t *`lv_theme_get_font_large`(lv_obj_t *obj)

Get the subtitle font of the theme

Parametri

obj -- pointer to an object

Ritorna

pointer to the font

lv_color_t `lv_theme_get_color_primary`(lv_obj_t *obj)

Get the primary color of the theme

Parametri

obj -- pointer to an object

Ritorna
the color

lv_color_t **lv_theme_get_color_secondary**(lv_obj_t *obj)

Get the secondary color of the theme

Parametri
obj -- pointer to an object

Ritorna
the color

14.22.2 lv_theme_private.h

struct **_lv_theme_t**

Public Members

lv_theme_apply_cb_t **apply_cb**

lv_theme_t ***parent**

Apply the current theme's style on top of this theme.

void ***user_data**

lv_display_t ***disp**

lv_color_t **color_primary**

lv_color_t **color_secondary**

const lv_font_t ***font_small**

const lv_font_t ***font_normal**

const lv_font_t ***font_large**

uint32_t **flags**

Any custom flag used by the theme

14.22.3 default

lv_theme_default.h

Functions

lv_theme_t ***lv_theme_default_init**(lv_display_t *disp, *lv_color_t* color_primary, *lv_color_t* color_secondary, bool dark, const lv_font_t *font)

Initialize the theme

Parametri

- **disp** -- pointer to display
- **color_primary** -- the primary color of the theme
- **color_secondary** -- the secondary color for the theme
- **dark** --

- **font** -- pointer to a font to use.

Ritorna
a pointer to reference this theme later

bool **lv_theme_default_is_inited**(void)

Check if default theme is initialized

Ritorna
true if default theme is initialized, false otherwise

lv_theme_t ***lv_theme_default_get**(void)

Get default theme

Ritorna
a pointer to default theme, or NULL if this is not initialized

void **lv_theme_default_deinit**(void)

Deinitialize the default theme

14.22.4 mono

lv_theme_mono.h

Functions

lv_theme_t ***lv_theme_mono_init**(lv_display_t *disp, bool dark_bg, const lv_font_t *font)

Initialize the theme

Parametri
- **disp** -- pointer to display
- **dark_bg** --
- **font** -- pointer to a font to use.

Ritorna
a pointer to reference this theme later

bool **lv_theme_mono_is_inited**(void)

Check if the theme is initialized

Ritorna
true if default theme is initialized, false otherwise

lv_theme_t ***lv_theme_mono_get**(void)

Get mono theme

Ritorna
a pointer to mono theme, or NULL if this is not initialized

void **lv_theme_mono_deinit**(void)

Deinitialize the mono theme

14.22.5 simple

lv_theme_simple.h

Functions

lv_theme_t ***lv_theme_simple_init**(lv_display_t *disp)

Initialize the theme

Parametri
disp -- pointer to display

Ritorna
a pointer to reference this theme later

bool **lv_theme_simple_is_inited**(void)

Check if the theme is initialized

Ritorna
true if default theme is initialized, false otherwise

lv_theme_t ***lv_theme_simple_get**(void)

Get simple theme

Ritorna
a pointer to simple theme, or NULL if this is not initialized

void **lv_theme_simple_deinit**(void)

Deinitialize the simple theme

14.23 tick

14.23.1 lv_tick.h

Provide access to the system tick with 1 millisecond resolution

Typedef

typedef uint32_t (***lv_tick_get_cb_t**)(void)

typedef void (***lv_delay_cb_t**)(uint32_t ms)

Functions

void **lv_tick_inc**(uint32_t tick_period)

You have to call this function periodically

Parametri
tick_period -- the call period of this function in milliseconds

uint32_t **lv_tick_get**(void)

Get the elapsed milliseconds since start up

Ritorna
the elapsed milliseconds

uint32_t **lv_tick_elaps**(uint32_t prev_tick)

Get the elapsed milliseconds since a previous time stamp

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

Ritorna
the elapsed milliseconds since 'prev_tick'

uint32_t **lv_tick_diff**(uint32_t tick, uint32_t prev_tick)

Get the elapsed milliseconds between two time stamps

Parametri
- **tick** -- a time stamp
- **prev_tick** -- a time stamp before `tick`

Ritorna
the elapsed milliseconds between `prev_tick` and `tick`

void **lv_delay_ms**(uint32_t ms)

Delay for the given milliseconds. By default it's a blocking delay, but with *`lv_delay_set_cb()`* a custom delay function can be set too

Parametri
ms -- the number of milliseconds to delay

void **lv_delay_set_cb**(*lv_delay_cb_t* cb)

Set a callback for a blocking delay

Parametri

cb -- pointer to a callback

void **lv_tick_set_cb**(*lv_tick_get_cb_t* cb)

Set the custom callback for 'lv_tick_get'

Parametri

cb -- call this callback on 'lv_tick_get'

lv_tick_get_cb_t **lv_tick_get_cb**(void)

Get the custom callback for 'lv_tick_get'

Ritorna

call this callback on 'lv_tick_get'

14.23.2 lv_tick_private.h

struct **lv_tick_state_t**

Public Members

uint32_t **sys_time**

volatile uint8_t **sys_irq_flag**

lv_tick_get_cb_t **tick_get_cb**

lv_delay_cb_t **delay_cb**

14.24 widgets

14.24.1 3dtexture

lv_3dtexture.h

Functions

lv_obj_t ***lv_3dtexture_create**(lv_obj_t *parent)

Create a 3dtexture object

Parametri

parent -- pointer to an object, it will be the parent of the new 3dtexture

Ritorna

pointer to the created 3dtexture

void **lv_3dtexture_set_src**(lv_obj_t *obj, lv_3dtexture_id_t id)

Set the source texture of the widget. The object size should be manually set to match.

Parametri

- **obj** -- the 3dtexture widget
- **id** -- the texture handle from the 3D graphics backend. I.e., an `unsigned int` texture for OpenGL.

Variables

const lv_obj_class_t **lv_3dtexture_class**

lv_3dtexture_private.h

struct **_lv_3dtexture_t**

Public Members

lv_obj_t **obj**

lv_3dtexture_id_t **id**

14.24.2 animimage

lv_animimage.h

Enums

enum **_lv_property_animimage_id_t**

Modified by NXP in 2025

Values:

enumerator **LV_PROPERTY_ANIMIMAGE_SRC**

enumerator **LV_PROPERTY_ANIMIMAGE_DURATION**

enumerator **LV_PROPERTY_ANIMIMAGE_REPEAT_COUNT**

enumerator **LV_PROPERTY_ANIMIMAGE_SRC_COUNT**

enumerator **LV_PROPERTY_ANIMIMAGE_END**

enum **lv_animimg_part_t**

Image parts

Values:

enumerator **LV_ANIM_IMAGE_PART_MAIN**

Functions

lv_obj_t ***lv_animimg_create**(lv_obj_t *parent)

Create an animation image objects

Parametri

parent -- pointer to an object, it will be the parent of the new button

Ritorna

pointer to the created animation image object

void **lv_animimg_set_src**(lv_obj_t *obj, const void *dsc[], size_t num)

Set the image animation images source.

Parametri

- **obj** -- pointer to an animation image object
- **dsc** -- pointer to a series images
- **num** -- images' number

void **lv_animimg_set_src_reverse**(lv_obj_t *obj, const void *dsc[], size_t num)

Set the images source for flip playback of animation image.

Parametri

- **obj** -- pointer to an animation image object
- **dsc** -- pointer to a series images
- **num** -- images' number

void **lv_animimg_start**(lv_obj_t *obj)

Startup the image animation.

Parametri

obj -- pointer to an animation image object

bool **lv_animimg_delete**(lv_obj_t *obj)

Delete the image animation.

Parametri

obj -- pointer to an animation image object

void **lv_animimg_set_duration**(lv_obj_t *obj, uint32_t duration)

Set the image animation duration time. unit:ms

Parametri

- **obj** -- pointer to an animation image object
- **duration** -- the duration in milliseconds

void **lv_animimg_set_repeat_count**(lv_obj_t *obj, uint32_t count)

Set the image animation repeatedly play times.

Parametri

- **obj** -- pointer to an animation image object
- **count** -- the number of times to repeat the animation

void **lv_animimg_set_reverse_duration**(lv_obj_t *obj, uint32_t duration)

Make the image animation to play back to when the forward direction is ready.

Parametri

- **obj** -- pointer to an animation image object
- **duration** -- the duration of the playback image animation in milliseconds. 0: disable playback

void **lv_animimg_set_reverse_delay**(lv_obj_t *obj, uint32_t duration)

Make the image animation to play back to when the forward direction is ready.

Parametri

- **obj** -- pointer to an animation image object
- **duration** -- delay in milliseconds before starting the playback image animation.

void **lv_animimg_set_start_cb**(lv_obj_t *obj, *lv_anim_start_cb_t* start_cb)

Set a function call when the animation image really starts (considering `delay`)

Parametri

- **obj** -- pointer to an animation image object
- **start_cb** -- a function call when the animation is start

void **lv_animimg_set_completed_cb**(lv_obj_t *obj, *lv_anim_completed_cb_t* completed_cb)

Set a function call when the animation is completed

Parametri

- **obj** -- pointer to an animation image object
- **completed_cb** -- a function call when the animation is completed

const void **`lv_animimg_get_src`(lv_obj_t *obj)

Get the image animation images source.

Parametri

obj -- pointer to an animation image object

Ritorna

a pointer that will point to a series images

uint8_t `lv_animimg_get_src_count`(lv_obj_t *obj)

Get the image animation images source.

Parametri

obj -- pointer to an animation image object

Ritorna

the number of source images

uint32_t `lv_animimg_get_duration`(lv_obj_t *obj)

Get the image animation duration time. unit:ms

Parametri

obj -- pointer to an animation image object

Ritorna

the animation duration time

uint32_t `lv_animimg_get_repeat_count`(lv_obj_t *obj)

Get the image animation repeat play times.

Parametri

obj -- pointer to an animation image object

Ritorna

the repeat count

lv_anim_t *`lv_animimg_get_anim`(lv_obj_t *obj)

Get the image animation underlying animation.

Parametri

obj -- pointer to an animation image object

Ritorna

the animation reference

Variables

const lv_obj_class_t `lv_animimg_class`

lv_animimage_private.h

struct `_lv_animimg_t`

#include <lv_animimage_private.h> Data of the animimage

Public Members

lv_image_t `img`

lv_anim_t `anim`

const void **`dsc`

int8_t `pic_count`

14.24.3 arc

lv_arc.h

Enums

enum **lv_arc_mode_t**

Values:

enumerator **LV_ARC_MODE_NORMAL**

enumerator **LV_ARC_MODE_SYMMETRICAL**

enumerator **LV_ARC_MODE_REVERSE**

Functions

lv_obj_t ***lv_arc_create**(lv_obj_t *parent)

Create an arc object

Parametri

parent -- pointer to an object, it will be the parent of the new arc

Ritorna

pointer to the created arc

void **lv_arc_set_start_angle**(lv_obj_t *obj, lv_value_precise_t start)

Set the start angle of an arc. 0 deg: right, 90 bottom, etc.

Parametri

- **obj** -- pointer to an arc object
- **start** -- the start angle. (if `LV_USE_FLOAT` is enabled it can be fractional too.)

void **lv_arc_set_end_angle**(lv_obj_t *obj, lv_value_precise_t end)

Set the end angle of an arc. 0 deg: right, 90 bottom, etc.

Parametri

- **obj** -- pointer to an arc object
- **end** -- the end angle (if `LV_USE_FLOAT` is enabled it can be fractional too.)

void **lv_arc_set_angles**(lv_obj_t *obj, lv_value_precise_t start, lv_value_precise_t end)

Set the start and end angles

Parametri

- **obj** -- pointer to an arc object
- **start** -- the start angle (if `LV_USE_FLOAT` is enabled it can be fractional too.)
- **end** -- the end angle (if `LV_USE_FLOAT` is enabled it can be fractional too.)

void **lv_arc_set_bg_start_angle**(lv_obj_t *obj, lv_value_precise_t start)

Set the start angle of an arc background. 0 deg: right, 90 bottom, etc.

Parametri

- **obj** -- pointer to an arc object
- **start** -- the start angle (if `LV_USE_FLOAT` is enabled it can be fractional too.)

void **lv_arc_set_bg_end_angle**(lv_obj_t *obj, lv_value_precise_t end)

Set the start angle of an arc background. 0 deg: right, 90 bottom etc.

Parametri

- **obj** -- pointer to an arc object
- **end** -- the end angle (if `LV_USE_FLOAT` is enabled it can be fractional too.)

void **lv_arc_set_bg_angles**(lv_obj_t *obj, lv_value_precise_t start, lv_value_precise_t end)

Set the start and end angles of the arc background

Parametri

- **obj** -- pointer to an arc object
- **start** -- the start angle (if LV_USE_FLOAT is enabled it can be fractional too.)
- **end** -- the end angle (if LV_USE_FLOAT is enabled it can be fractional too.)

void **lv_arc_set_rotation**(lv_obj_t *obj, int32_t rotation)

Set the rotation for the whole arc

Parametri

- **obj** -- pointer to an arc object
- **rotation** -- rotation angle

void **lv_arc_set_mode**(lv_obj_t *obj, *lv_arc_mode_t* type)

Set the type of arc.

Parametri

- **obj** -- pointer to arc object
- **type** -- arc's mode

void **lv_arc_set_value**(lv_obj_t *obj, int32_t value)

Set a new value on the arc

Parametri

- **obj** -- pointer to an arc object
- **value** -- new value

void **lv_arc_set_range**(lv_obj_t *obj, int32_t min, int32_t max)

Set minimum and the maximum values of an arc

Parametri

- **obj** -- pointer to the arc object
- **min** -- minimum value
- **max** -- maximum value

void **lv_arc_set_min_value**(lv_obj_t *obj, int32_t min)

Set the minimum values of an arc

Parametri

- **obj** -- pointer to the arc object
- **min** -- minimum value

void **lv_arc_set_max_value**(lv_obj_t *obj, int32_t max)

Set the maximum values of an arc

Parametri

- **obj** -- pointer to the arc object
- **max** -- maximum value

void **lv_arc_set_change_rate**(lv_obj_t *obj, uint32_t rate)

Set a change rate to limit the speed how fast the arc should reach the pressed point.

Parametri

- **obj** -- pointer to an arc object
- **rate** -- the change rate

void **lv_arc_set_knob_offset**(lv_obj_t *obj, int32_t offset)

Set an offset angle for the knob

Parametri

- **obj** -- pointer to an arc object
- **offset** -- knob offset from main arc in degrees

lv_value_precise_t **lv_arc_get_angle_start**(lv_obj_t *obj)

Get the start angle of an arc.

Parametri

obj -- pointer to an arc object

Ritorna

the start angle [0..360] (if LV_USE_FLOAT is enabled it can be fractional too.)

lv_value_precise_t **lv_arc_get_angle_end**(lv_obj_t *obj)

Get the end angle of an arc.

Parametri

obj -- pointer to an arc object

Ritorna

the end angle [0..360] (if LV_USE_FLOAT is enabled it can be fractional too.)

lv_value_precise_t **lv_arc_get_bg_angle_start**(lv_obj_t *obj)

Get the start angle of an arc background.

Parametri

obj -- pointer to an arc object

Ritorna

the start angle [0..360] (if LV_USE_FLOAT is enabled it can be fractional too.)

lv_value_precise_t **lv_arc_get_bg_angle_end**(lv_obj_t *obj)

Get the end angle of an arc background.

Parametri

obj -- pointer to an arc object

Ritorna

the end angle [0..360] (if LV_USE_FLOAT is enabled it can be fractional too.)

int32_t **lv_arc_get_value**(const lv_obj_t *obj)

Get the value of an arc

Parametri

obj -- pointer to an arc object

Ritorna

the value of the arc

int32_t **lv_arc_get_min_value**(const lv_obj_t *obj)

Get the minimum value of an arc

Parametri

obj -- pointer to an arc object

Ritorna

the minimum value of the arc

int32_t **lv_arc_get_max_value**(const lv_obj_t *obj)

Get the maximum value of an arc

Parametri

obj -- pointer to an arc object

Ritorna

the maximum value of the arc

lv_arc_mode_t **lv_arc_get_mode**(const lv_obj_t *obj)

Get whether the arc is type or not.

Parametri

obj -- pointer to an arc object

Ritorna

arc's mode

int32_t **lv_arc_get_rotation**(const lv_obj_t *obj)

Get the rotation for the whole arc

Parametri

obj -- pointer to an arc object

Ritorna

arc's current rotation

int32_t **lv_arc_get_knob_offset**(const lv_obj_t *obj)

Get the current knob angle offset

Parametri

obj -- pointer to an arc object

Ritorna
arc's current knob offset

void **lv_arc_align_obj_to_angle**(const lv_obj_t *obj, lv_obj_t *obj_to_align, int32_t r_offset)

Align an object to the current position of the arc (knob)

Parametri
- **obj** -- pointer to an arc object
- **obj_to_align** -- pointer to an object to align
- **r_offset** -- consider the radius larger with this value (< 0: for smaller radius)

void **lv_arc_rotate_obj_to_angle**(const lv_obj_t *obj, lv_obj_t *obj_to_rotate, int32_t r_offset)

Rotate an object to the current position of the arc (knob)

Parametri
- **obj** -- pointer to an arc object
- **obj_to_rotate** -- pointer to an object to rotate
- **r_offset** -- consider the radius larger with this value (< 0: for smaller radius)

Variables

const lv_obj_class_t **lv_arc_class**

lv_arc_private.h

struct **_lv_arc_t**

Public Members

lv_obj_t **obj**

int32_t **rotation**

lv_value_precise_t **indic_angle_start**

lv_value_precise_t **indic_angle_end**

lv_value_precise_t **bg_angle_start**

lv_value_precise_t **bg_angle_end**

int32_t **value**

Current value of the arc

int32_t **min_value**

Minimum value of the arc

int32_t **max_value**

Maximum value of the arc

uint32_t **dragging**

uint32_t **type**

uint32_t **min_close**

1: the last pressed angle was closer to minimum end

uint32_t **in_out**

1: The click was within the background arc angles. 0: Click outside

uint32_t **chg_rate**

Drag angle rate of change of the arc (degrees/sec)

uint32_t **last_tick**

Last dragging event timestamp of the arc

lv_value_precise_t **last_angle**

Last dragging angle of the arc

int16_t **knob_offset**

knob offset from the main arc

14.24.4 arclabel

lv_arclabel.h

Defines

LV_ARCLABEL_DOT_NUM

LV_ARCLABEL_DEFAULT_TEXT

Enums

enum **lv_arclabel_dir_t**

Values:

enumerator **LV_ARCLABEL_DIR_CLOCKWISE**

enumerator **LV_ARCLABEL_DIR_COUNTER_CLOCKWISE**

enum **lv_arclabel_text_align_t**

Values:

enumerator **LV_ARCLABEL_TEXT_ALIGN_DEFAULT**

enumerator **LV_ARCLABEL_TEXT_ALIGN_LEADING**

enumerator **LV_ARCLABEL_TEXT_ALIGN_CENTER**

enumerator **LV_ARCLABEL_TEXT_ALIGN_TRAILING**

Functions

lv_obj_t ***lv_arclabel_create**(lv_obj_t *parent)

Create an arc label object

Parametri

parent -- pointer to an object, it will be the parent of the new arc label

Ritorna

pointer to the created arc label

void **lv_arclabel_set_text**(lv_obj_t *obj, const char *text)

Set the text of the arc label.

This function sets the text displayed by an arc label object.

Parametri

- **obj** -- Pointer to the arc label object.
- **text** -- Pointer to a null-terminated string containing the new text for the label.

void **lv_arclabel_set_text_fmt**(lv_obj_t *obj, const char *fmt, ...)

Set the formatted text of an arc label object.

This function sets the text of an arc label object with support for variable arguments formatting, similar to `printf`.

Parametri

- **obj** -- The arc label object to set the text for.
- **fmt** -- A format string that specifies how subsequent arguments are converted to text.
- **...** -- Arguments following the format string that are used to replace format specifiers in the format string.

void **lv_arclabel_set_text_static**(lv_obj_t *obj, const char *text)

Sets a new static text for the arc label or refreshes it with the current text. The 'text' must remain valid in memory; the arc label does not manage its lifecycle.

Parametri

- **obj** -- Pointer to the arc label object.
- **text** -- Pointer to the new text. If NULL, the label is refreshed with its current text.

void **lv_arclabel_set_angle_start**(lv_obj_t *obj, lv_value_precise_t start)

Set the start angle of an arc. 0 deg: right, 90 bottom, etc.

Parametri

- **obj** -- pointer to an arc label object
- **start** -- the start angle. (if `LV_USE_FLOAT` is enabled it can be fractional too.)

void **lv_arclabel_set_angle_size**(lv_obj_t *obj, lv_value_precise_t size)

Set the end angle of an arc. 0 deg: right, 90 bottom, etc.

Parametri

- **obj** -- pointer to an arc label object
- **size** -- the angle size (if `LV_USE_FLOAT` is enabled it can be fractional too.)

void **lv_arclabel_set_offset**(lv_obj_t *obj, int32_t offset)

Set the rotation for the whole arc

Parametri

- **obj** -- pointer to an arc label object
- **offset** -- rotation angle

void **lv_arclabel_set_dir**(lv_obj_t *obj, *lv_arclabel_dir_t* dir)

Set the type of arc.

Parametri

- **obj** -- pointer to and arc label object
- **dir** -- arc label's direction

void **lv_arclabel_set_recolor**(lv_obj_t *obj, bool en)

Enable the recoloring by in-line commands

Parametri

- **obj** -- pointer to an arc label object
- **en** -- true: enable recoloring, false: disable Example: "This is a #ff0000 red# word"

void **lv_arclabel_set_radius**(lv_obj_t *obj, uint32_t radius)

Set the radius for an arc label object.

Parametri

- **obj** -- pointer to the arc label object.
- **radius** -- The radius value to set for the label's curvature, in pixels.

void **lv_arclabel_set_center_offset_x**(lv_obj_t *obj, uint32_t x)

Set the center offset x for an arc label object.

Parametri

- **obj** -- pointer to an arc label object
- **x** -- the x offset

void **lv_arclabel_set_center_offset_y**(lv_obj_t *obj, uint32_t y)

Set the center offset y for an arc label object.

Parametri

- **obj** -- pointer to an arc label object
- **y** -- the y offset

void **lv_arclabel_set_text_vertical_align**(lv_obj_t *obj, *lv_arclabel_text_align_t* align)

Set the text vertical alignment for an arc label object.

Parametri

- **obj** -- pointer to an arc label object
- **align** -- the vertical alignment

void **lv_arclabel_set_text_horizontal_align**(lv_obj_t *obj, *lv_arclabel_text_align_t* align)

Set the text horizontal alignment for an arc label object.

Parametri

- **obj** -- pointer to an arc label object
- **align** -- the horizontal alignment

lv_value_precise_t **lv_arclabel_get_angle_start**(lv_obj_t *obj)

Get the start angle of an arc label.

Parametri

obj -- pointer to an arc label object

Ritorna

the start angle [0..360] (if `LV_USE_FLOAT` is enabled it can be fractional too.)

lv_value_precise_t **lv_arclabel_get_angle_size**(lv_obj_t *obj)

Get the angle size of an arc label.

Parametri

obj -- pointer to an arc label object

Ritorna

the end angle [0..360] (if `LV_USE_FLOAT` is enabled it can be fractional too.)

lv_arclabel_dir_t **lv_arclabel_get_dir**(const lv_obj_t *obj)

Get whether the arc label is type or not.

Parametri

obj -- pointer to an arc label object

Ritorna

arc label's direction

bool **lv_arclabel_get_recolor**(lv_obj_t *obj)

Enable the recoloring by in-line commands

Vedi anche

lv_arclabel_set_recolor

Parametri
obj -- pointer to a label object
Ritorna
true: enable recoloring, false: disable

uint32_t **lv_arclabel_get_radius**(lv_obj_t *obj)

Get the text of the arc label.
Parametri
obj -- pointer to an arc label object
Ritorna
the radius of the arc label

uint32_t **lv_arclabel_get_center_offset_x**(lv_obj_t *obj)

Get the center offset x for an arc label object.
Parametri
obj -- pointer to an arc label object
Ritorna
the x offset

uint32_t **lv_arclabel_get_center_offset_y**(lv_obj_t *obj)

Get the center offset y for an arc label object.
Parametri
obj -- pointer to an arc label object
Ritorna
the y offset

lv_arclabel_text_align_t **lv_arclabel_get_text_vertical_align**(lv_obj_t *obj)

Get the text vertical alignment for an arc label object.
Parametri
obj -- pointer to an arc label object
Ritorna
the vertical alignment

lv_arclabel_text_align_t **lv_arclabel_get_text_horizontal_align**(lv_obj_t *obj)

Get the text horizontal alignment for an arc label object.
Parametri
obj -- pointer to an arc label object
Ritorna
the horizontal alignment

Variables

const lv_obj_class_t **lv_arclabel_class**

lv_arclabel_private.h

struct **_lv_arclabel_t**

Public Members

lv_obj_t **obj**

char ***text**

char **dot**[3 + 1]

Bytes that have been replaced with dots

uint32_t **dot_begin**

Offset where bytes have been replaced with dots

lv_value_precise_t **angle_start**

The starting angle of the arc in degrees.

This variable represents the beginning of the arc's angular range in terms of degrees. It is used in conjunction with `angle_size` to define the section of the circle that the arc covers. Values are normalized to the range [0, 360), ensuring compatibility with circular representations.

Modifying this value affects the visual rendering of the arc and its associated indicators or labels, necessitating a subsequent invalidation or update of the object's display to reflect the changes.

> **Nota**
>
> When setting this value programmatically, ensure it does not exceed 360 degrees, as it will be automatically adjusted to fit within the valid range.

lv_value_precise_t **angle_size**

uint32_t **offset**

uint32_t **radius**

lv_point_t **center_offset**

lv_arclabel_dir_t **dir**

lv_arclabel_text_align_t **text_align_v**

Vertical text alignment

lv_arclabel_text_align_t **text_align_h**

Horizontal text alignment

uint8_t **static_txt**

Flag to indicate the text is static

uint8_t **recolor**

Enable in-line letter re-coloring

14.24.5 bar

lv_bar.h

Enums

enum **lv_bar_mode_t**

Values:

enumerator **LV_BAR_MODE_NORMAL**

enumerator **LV_BAR_MODE_SYMMETRICAL**

enumerator **LV_BAR_MODE_RANGE**

enum **lv_bar_orientation_t**

Values:

enumerator **LV_BAR_ORIENTATION_AUTO**

enumerator **LV_BAR_ORIENTATION_HORIZONTAL**

enumerator **LV_BAR_ORIENTATION_VERTICAL**

Functions

lv_obj_t ***lv_bar_create**(lv_obj_t *parent)

Create a bar object

Parametri

parent -- pointer to an object, it will be the parent of the new bar

Ritorna

pointer to the created bar

void **lv_bar_set_value**(lv_obj_t *obj, int32_t value, *lv_anim_enable_t* anim)

Set a new value on the bar

Parametri

- **obj** -- pointer to a bar object
- **value** -- new value
- **anim** -- LV_ANIM_ON: set the value with an animation; LV_ANIM_OFF: change the value immediately

void **lv_bar_set_start_value**(lv_obj_t *obj, int32_t start_value, *lv_anim_enable_t* anim)

Set a new start value on the bar

Parametri

- **obj** -- pointer to a bar object
- **start_value** -- new start value
- **anim** -- LV_ANIM_ON: set the value with an animation; LV_ANIM_OFF: change the value immediately

void **lv_bar_set_range**(lv_obj_t *obj, int32_t min, int32_t max)

Set minimum and the maximum values of a bar

> **Nota**
>
> If min is greater than max, the drawing direction becomes to the opposite direction.

Parametri
- **obj** -- pointer to the bar object
- **min** -- minimum value
- **max** -- maximum value

void **lv_bar_set_min_value**(lv_obj_t *obj, int32_t min)

Set minimum value of a bar

Parametri
- **obj** -- pointer to the bar object
- **min** -- minimum value

void **lv_bar_set_max_value**(lv_obj_t *obj, int32_t max)

Set maximum value of a bar

Parametri
- **obj** -- pointer to the bar object
- **max** -- maximum value

void **lv_bar_set_mode**(lv_obj_t *obj, *lv_bar_mode_t* mode)

Set the type of bar.

Parametri
- **obj** -- pointer to bar object
- **mode** -- bar type from `lv_bar_mode_t`

void **lv_bar_set_orientation**(lv_obj_t *obj, *lv_bar_orientation_t* orientation)

Set the orientation of bar.

Parametri
- **obj** -- pointer to bar object
- **orientation** -- bar orientation from `lv_bar_orientation_t`

int32_t **lv_bar_get_value**(const lv_obj_t *obj)

Get the value of a bar

Parametri
obj -- pointer to a bar object

Ritorna
the value of the bar

int32_t **lv_bar_get_start_value**(const lv_obj_t *obj)

Get the start value of a bar

Parametri
obj -- pointer to a bar object

Ritorna
the start value of the bar

int32_t **lv_bar_get_min_value**(const lv_obj_t *obj)

Get the minimum value of a bar

Parametri
obj -- pointer to a bar object

Ritorna
the minimum value of the bar

int32_t **lv_bar_get_max_value**(const lv_obj_t *obj)

Get the maximum value of a bar

Parametri
obj -- pointer to a bar object

Ritorna
the maximum value of the bar

lv_bar_mode_t **lv_bar_get_mode**(lv_obj_t *obj)

Get the type of bar.

Parametri
obj -- pointer to bar object

Ritorna
bar type from `lv_bar_mode_t`

lv_bar_orientation_t **lv_bar_get_orientation**(lv_obj_t *obj)

Get the orientation of bar.

Parametri
obj -- pointer to bar object

Ritorna
bar orientation from `lv_bar_orientation_t`

bool **lv_bar_is_symmetrical**(lv_obj_t *obj)

Give the bar is in symmetrical mode or not

Parametri
obj -- pointer to bar object

Ritorna
true: in symmetrical mode false : not in

Variables

const lv_obj_class_t **lv_bar_class**

lv_bar_private.h

struct **_lv_bar_anim_t**

Public Members

lv_obj_t ***bar**

int32_t **anim_start**

int32_t **anim_end**

int32_t **anim_state**

struct **_lv_bar_t**

Public Members

lv_obj_t **obj**

int32_t **cur_value**

Current value of the bar

int32_t **min_value**

Minimum value of the bar

int32_t **max_value**

Maximum value of the bar

int32_t **start_value**

Start value of the bar

lv_area_t **indic_area**

Save the indicator area. Might be used by derived types

bool **val_reversed**

Whether value been reversed

lv_bar_anim_t **cur_value_anim**

lv_bar_anim_t **start_value_anim**

lv_bar_mode_t **mode**

Type of bar

lv_bar_orientation_t **orientation**

Orientation of bar

14.24.6 button

lv_button.h

Functions

lv_obj_t ***lv_button_create**(lv_obj_t *parent)

Create a button object

Parametri

parent -- pointer to an object, it will be the parent of the new button

Ritorna

pointer to the created button

Variables

const lv_obj_class_t **lv_button_class**

lv_button_private.h

struct **_lv_button_t**

Public Members

lv_obj_t **obj**

14.24.7 buttonmatrix

lv_buttonmatrix.h

Defines

LV_BUTTONMATRIX_BUTTON_NONE

Typedef

typedef bool (*`lv_buttonmatrix_button_draw_cb_t`)(lv_obj_t *btnm, uint32_t btn_id, const *lv_area_t* *draw_area, const *lv_area_t* *clip_area)

Enums

enum `lv_buttonmatrix_ctrl_t`

Type to store button control flags (disabled, hidden etc.) The least-significant 4 bits are used to store button-width proportions in range [1..15].

Values:

enumerator `LV_BUTTONMATRIX_CTRL_NONE`

No extra control, use the default settings

enumerator `LV_BUTTONMATRIX_CTRL_WIDTH_1`

Set the width to 1 relative to the other buttons in the same row

enumerator `LV_BUTTONMATRIX_CTRL_WIDTH_2`

Set the width to 2 relative to the other buttons in the same row

enumerator `LV_BUTTONMATRIX_CTRL_WIDTH_3`

Set the width to 3 relative to the other buttons in the same row

enumerator `LV_BUTTONMATRIX_CTRL_WIDTH_4`

Set the width to 4 relative to the other buttons in the same row

enumerator `LV_BUTTONMATRIX_CTRL_WIDTH_5`

Set the width to 5 relative to the other buttons in the same row

enumerator `LV_BUTTONMATRIX_CTRL_WIDTH_6`

Set the width to 6 relative to the other buttons in the same row

enumerator `LV_BUTTONMATRIX_CTRL_WIDTH_7`

Set the width to 7 relative to the other buttons in the same row

enumerator `LV_BUTTONMATRIX_CTRL_WIDTH_8`

Set the width to 8 relative to the other buttons in the same row

enumerator `LV_BUTTONMATRIX_CTRL_WIDTH_9`

Set the width to 9 relative to the other buttons in the same row

enumerator `LV_BUTTONMATRIX_CTRL_WIDTH_10`

Set the width to 10 relative to the other buttons in the same row

enumerator `LV_BUTTONMATRIX_CTRL_WIDTH_11`

Set the width to 11 relative to the other buttons in the same row

enumerator `LV_BUTTONMATRIX_CTRL_WIDTH_12`

Set the width to 12 relative to the other buttons in the same row

enumerator **LV_BUTTONMATRIX_CTRL_WIDTH_13**

Set the width to 13 relative to the other buttons in the same row

enumerator **LV_BUTTONMATRIX_CTRL_WIDTH_14**

Set the width to 14 relative to the other buttons in the same row

enumerator **LV_BUTTONMATRIX_CTRL_WIDTH_15**

Set the width to 15 relative to the other buttons in the same row

enumerator **LV_BUTTONMATRIX_CTRL_HIDDEN**

Hides button; it continues to hold its space in layout.

enumerator **LV_BUTTONMATRIX_CTRL_NO_REPEAT**

Do not emit LV_EVENT_LONG_PRESSED_REPEAT events while button is long-pressed.

enumerator **LV_BUTTONMATRIX_CTRL_DISABLED**

Disables button like LV_STATE_DISABLED on normal Widgets.

enumerator **LV_BUTTONMATRIX_CTRL_CHECKABLE**

Enable toggling of LV_STATE_CHECKED when clicked.

enumerator **LV_BUTTONMATRIX_CTRL_CHECKED**

Make the button checked. It will use the :cpp:enumerator:`LV_STATE_CHECHKED` styles.

enumerator **LV_BUTTONMATRIX_CTRL_CLICK_TRIG**

1: Enables sending LV_EVENT_VALUE_CHANGE on CLICK, 0: sends LV_EVENT_VALUE_CHANGE on PRESS.

enumerator **LV_BUTTONMATRIX_CTRL_POPOVER**

Show button text in a pop-over while being pressed.

enumerator **LV_BUTTONMATRIX_CTRL_RECOLOR**

Enable text recoloring with `#color`

enumerator **LV_BUTTONMATRIX_CTRL_RESERVED_1**

Reserved for later use

enumerator **LV_BUTTONMATRIX_CTRL_RESERVED_2**

Reserved for later use

enumerator **LV_BUTTONMATRIX_CTRL_CUSTOM_1**

Custom free-to-use flag

enumerator **LV_BUTTONMATRIX_CTRL_CUSTOM_2**

Custom free-to-use flag

Functions

lv_obj_t *`lv_buttonmatrix_create`(lv_obj_t *parent)

Create a button matrix object

Parametri

parent -- pointer to an object, it will be the parent of the new button matrix

Ritorna

pointer to the created button matrix

void `lv_buttonmatrix_set_map`(lv_obj_t *obj, const char *const map[])

Set a new map. Buttons will be created/deleted according to the map. The button matrix keeps a reference to the map and so the string array must not be deallocated during the life of the matrix.

Parametri

- **obj** -- pointer to a button matrix object
- **map** -- pointer a string array. The last string has to be: "". Use "\n" to make a line break.

void `lv_buttonmatrix_set_ctrl_map`(lv_obj_t *obj, const *lv_buttonmatrix_ctrl_t* ctrl_map[])

Set the button control map (hidden, disabled etc.) for a button matrix. The control map array will be copied and so may be deallocated after this function returns.

Parametri

- **obj** -- pointer to a button matrix object
- **ctrl_map** -- pointer to an array of `lv_button_ctrl_t` control bytes. The length of the array and position of the elements must match the number and order of the individual buttons (i.e. excludes newline entries). An element of the map should look like e.g.: `ctrl_map[0] = width | LV_BUTTONMATRIX_CTRL_NO_REPEAT | LV_BUTTONMATRIX_CTRL_TGL_ENABLE`

void `lv_buttonmatrix_set_selected_button`(lv_obj_t *obj, uint32_t btn_id)

Set the selected buttons

Parametri

- **obj** -- pointer to button matrix object
- **btn_id** -- 0 based index of the button to modify. (Not counting new lines)

void `lv_buttonmatrix_set_button_ctrl`(lv_obj_t *obj, uint32_t btn_id, *lv_buttonmatrix_ctrl_t* ctrl)

Set the attributes of a button of the button matrix

Parametri

- **obj** -- pointer to button matrix object
- **btn_id** -- 0 based index of the button to modify. (Not counting new lines)
- **ctrl** -- OR-ed attributes. E.g. `LV_BUTTONMATRIX_CTRL_NO_REPEAT | LV_BUTTONMATRIX_CTRL_CHECKABLE`

void `lv_buttonmatrix_clear_button_ctrl`(lv_obj_t *obj, uint32_t btn_id, *lv_buttonmatrix_ctrl_t* ctrl)

Clear the attributes of a button of the button matrix

Parametri

- **obj** -- pointer to button matrix object
- **btn_id** -- 0 based index of the button to modify. (Not counting new lines)
- **ctrl** -- OR-ed attributes. E.g. `LV_BUTTONMATRIX_CTRL_NO_REPEAT | LV_BUTTONMATRIX_CTRL_CHECKABLE`

void `lv_buttonmatrix_set_button_ctrl_all`(lv_obj_t *obj, *lv_buttonmatrix_ctrl_t* ctrl)

Set attributes of all buttons of a button matrix

Parametri

- **obj** -- pointer to a button matrix object
- **ctrl** -- attribute(s) to set from *`lv_buttonmatrix_ctrl_t`*. Values can be ORed.

void `lv_buttonmatrix_clear_button_ctrl_all`(lv_obj_t *obj, *lv_buttonmatrix_ctrl_t* ctrl)

Clear the attributes of all buttons of a button matrix

Parametri

- **obj** -- pointer to a button matrix object
- **ctrl** -- attribute(s) to set from *lv_buttonmatrix_ctrl_t*. Values can be ORed.

void **lv_buttonmatrix_set_button_width**(lv_obj_t *obj, uint32_t btn_id, uint32_t width)

Set a single button's relative width. This method will cause the matrix be regenerated and is a relatively expensive operation. It is recommended that initial width be specified using `lv_buttonmatrix_set_ctrl_map` and this method only be used for dynamic changes.

Parametri

- **obj** -- pointer to button matrix object
- **btn_id** -- 0 based index of the button to modify.
- **width** -- relative width compared to the buttons in the same row. [1..15]

void **lv_buttonmatrix_set_one_checked**(lv_obj_t *obj, bool en)

Make the button matrix like a selector widget (only one button may be checked at a time). `LV_BUTTONMATRIX_CTRL_CHECKABLE` must be enabled on the buttons to be selected using `lv_buttonmatrix_set_ctrl()` or *lv_buttonmatrix_set_button_ctrl_all()*.

Parametri

- **obj** -- pointer to a button matrix object
- **en** -- whether "one check" mode is enabled

const char *const ***lv_buttonmatrix_get_map**(const lv_obj_t *obj)

Get the current map of a button matrix

Parametri

obj -- pointer to a button matrix object

Ritorna

the current map

uint32_t **lv_buttonmatrix_get_selected_button**(const lv_obj_t *obj)

Get the index of the lastly "activated" button by the user (pressed, released, focused etc) Useful in the `event_cb` to get the text of the button, check if hidden etc.

Parametri

obj -- pointer to button matrix object

Ritorna

index of the last released button (LV_BUTTONMATRIX_BUTTON_NONE: if unset)

const char ***lv_buttonmatrix_get_button_text**(const lv_obj_t *obj, uint32_t btn_id)

Get the button's text

Parametri

- **obj** -- pointer to button matrix object
- **btn_id** -- the index a button not counting new line characters.

Ritorna

text of btn_index` button

bool **lv_buttonmatrix_has_button_ctrl**(lv_obj_t *obj, uint32_t btn_id, *lv_buttonmatrix_ctrl_t* ctrl)

Get the whether a control value is enabled or disabled for button of a button matrix

Parametri

- **obj** -- pointer to a button matrix object
- **btn_id** -- the index of a button not counting new line characters.
- **ctrl** -- control values to check (ORed value can be used)

Ritorna

true: the control attribute is enabled false: disabled

bool **lv_buttonmatrix_get_one_checked**(const lv_obj_t *obj)

Tell whether "one check" mode is enabled or not.

Parametri

obj -- Button matrix object

Ritorna

true: "one check" mode is enabled; false: disabled

Variables

const lv_obj_class_t **lv_buttonmatrix_class**

lv_buttonmatrix_private.h

struct **_lv_buttonmatrix_t**

#include <lv_buttonmatrix_private.h> Data of button matrix

Public Members

lv_obj_t **obj**

const char *const ***map_p**

Pointer to the current map

lv_area_t ***button_areas**

Array of areas of buttons

lv_buttonmatrix_ctrl_t ***ctrl_bits**

Array of control bytes

uint32_t **btn_cnt**

Number of button in 'map_p'(Handled by the library)

uint32_t **row_cnt**

Number of rows in 'map_p'(Handled by the library)

uint32_t **btn_id_sel**

Index of the active button (being pressed/released etc) or LV_BUTTONMATRIX_BUTTON_NONE

uint32_t **one_check**

1: Single button toggled at once

uint32_t **auto_free_map**

1: Automatically free the map when the widget is deleted

14.24.8 calendar

lv_calendar.h

Functions

lv_obj_t ***lv_calendar_create**(lv_obj_t *parent)

Create a calendar widget

Parametri

parent -- pointer to an object, it will be the parent of the new calendar

Ritorna

pointer the created calendar

void **lv_calendar_set_today_date**(lv_obj_t *obj, uint32_t year, uint32_t month, uint32_t day)

Set the today's year, month and day at once

Parametri

- **obj** -- pointer to a calendar object
- **year** -- today's year
- **month** -- today's month [1..12]
- **day** -- today's day [1..31]

void **lv_calendar_set_today_year**(lv_obj_t *obj, uint32_t year)

Set the today's year

Parametri

- **obj** -- pointer to a calendar object
- **year** -- today's year

void **lv_calendar_set_today_month**(lv_obj_t *obj, uint32_t month)

Set the today's year

Parametri

- **obj** -- pointer to a calendar object
- **month** -- today's month [1..12]

void **lv_calendar_set_today_day**(lv_obj_t *obj, uint32_t day)

Set the today's year

Parametri

- **obj** -- pointer to a calendar object
- **day** -- today's day [1..31]

void **lv_calendar_set_month_shown**(lv_obj_t *obj, uint32_t year, uint32_t month)

Set the currently shown year and month at once

Parametri

- **obj** -- pointer to a calendar object
- **year** -- shown year
- **month** -- shown month [1..12]

void **lv_calendar_set_shown_year**(lv_obj_t *obj, uint32_t year)

Set the currently shown year

Parametri

- **obj** -- pointer to a calendar object
- **year** -- shown year

void **lv_calendar_set_shown_month**(lv_obj_t *obj, uint32_t month)

Set the currently shown month

Parametri

- **obj** -- pointer to a calendar object
- **month** -- shown month [1..12]

void **lv_calendar_set_highlighted_dates**(lv_obj_t *obj, *lv_calendar_date_t* highlighted[], size_t date_num)

Set the highlighted dates

Parametri

- **obj** -- pointer to a calendar object
- **highlighted** -- pointer to an *lv_calendar_date_t* array containing the dates. Only the pointer will be saved so this variable can't be local which will be destroyed later.
- **date_num** -- number of dates in the array

void **lv_calendar_set_day_names**(lv_obj_t *obj, const char **day_names)

Set the name of the days

Parametri

- **obj** -- pointer to a calendar object

- **day_names** -- pointer to an array with the names. E.g. `const char * days[7] = {"Sun", "Mon", ...}` Only the pointer will be saved so this variable can't be local which will be destroyed later.

lv_obj_t ***lv_calendar_get_btnmatrix**(const lv_obj_t *obj)

Get the button matrix object of the calendar. It shows the dates and day names.

Parametri

obj -- pointer to a calendar object

Ritorna

pointer to a the button matrix

const *lv_calendar_date_t* ***lv_calendar_get_today_date**(const lv_obj_t *calendar)

Get the today's date

Parametri

calendar -- pointer to a calendar object

Ritorna

return pointer to an `lv_calendar_date_t` variable containing the date of today.

const *lv_calendar_date_t* ***lv_calendar_get_showed_date**(const lv_obj_t *calendar)

Get the currently showed

Parametri

calendar -- pointer to a calendar object

Ritorna

pointer to an `lv_calendar_date_t` variable containing the date is being shown.

lv_calendar_date_t ***lv_calendar_get_highlighted_dates**(const lv_obj_t *calendar)

Get the highlighted dates

Parametri

calendar -- pointer to a calendar object

Ritorna

pointer to an `lv_calendar_date_t` array containing the dates.

size_t **lv_calendar_get_highlighted_dates_num**(const lv_obj_t *calendar)

Get the number of the highlighted dates

Parametri

calendar -- pointer to a calendar object

Ritorna

number of highlighted days

lv_result_t **lv_calendar_get_pressed_date**(const lv_obj_t *calendar, *lv_calendar_date_t* *date)

Get the currently pressed day

Parametri

- **calendar** -- pointer to a calendar object
- **date** -- store the pressed date here

Ritorna

LV_RESULT_OK: there is a valid pressed date LV_RESULT_INVALID: there is no pressed data

Variables

const lv_obj_class_t **lv_calendar_class**

struct **lv_calendar_date_t**

#include <lv_calendar.h> Represents a date on the calendar object (platform-agnostic).

Public Members

uint16_t **year**

uint8_t **month**

1..12

uint8_t **day**

1..31

lv_calendar_chinese.h

Functions

void **lv_calendar_set_chinese_mode**(lv_obj_t *obj, bool en)

Enable the chinese calendar.

Parametri

- **obj** -- pointer to a calendar object.
- **en** -- true: enable chinese calendar; false: disable

const char ***lv_calendar_get_day_name**(*lv_calendar_date_t* *gregorian)

Get the name of the day

Parametri

gregorian -- to obtain the gregorian time for the name

Ritorna

return the name of the day

void **lv_calendar_gregorian_to_chinese**(*lv_calendar_date_t* *gregorian_time, *lv_calendar_chinese_t* *chinese_time)

Get the chinese time of the gregorian time (reference: https://www.cnblogs.com/liyang31tg/p/4123171.html)

Parametri

- **gregorian_time** -- need to convert to chinese time in gregorian time
- **chinese_time** -- the chinese time convert from gregorian time

struct **lv_calendar_chinese_t**

Public Members

lv_calendar_date_t **today**

bool **leep_month**

lv_calendar_header_arrow.h

Functions

lv_obj_t ***lv_calendar_add_header_arrow**(lv_obj_t *parent)

Create a calendar header with drop-drowns to select the year and month

Parametri

parent -- pointer to a calendar object.

Ritorna

the created header

Variables

const lv_obj_class_t **lv_calendar_header_arrow_class**

lv_calendar_header_dropdown.h

Functions

lv_obj_t ***lv_calendar_add_header_dropdown**(lv_obj_t *parent)

Create a calendar header with drop-drowns to select the year and month

Parametri

parent -- pointer to a calendar object.

Ritorna

the created header

void **lv_calendar_header_dropdown_set_year_list**(lv_obj_t *parent, const char *years_list)

Sets a custom calendar year list

Parametri

- **parent** -- pointer to a calendar object
- **years_list** -- pointer to an const char array with the years list, see lv_dropdown set_options for more information. E.g. `const char * years = "2023\n2022\n2021\n2020\n2019" Only the pointer will be saved so this variable can't be local which will be destroyed later.

Variables

const lv_obj_class_t **lv_calendar_header_dropdown_class**

lv_calendar_private.h

struct **_lv_calendar_t**

#include <lv_calendar_private.h> Data of calendar

Public Members

lv_obj_t **obj**

lv_obj_t ***btnm**

lv_calendar_date_t **today**

Date of today

lv_calendar_date_t **showed_date**

Currently visible month (day is ignored)

lv_calendar_date_t ***highlighted_dates**

Apply different style on these days (pointer to user-defined array)

size_t **highlighted_dates_num**

Number of elements in `highlighted_days`

const char ***map**[8 * 7]

bool **use_chinese_calendar**

char **nums**[7 * 6][20]

7 * 6: A week has 7 days, and the calendar displays 6 weeks in total. 20: Including the number of dates, line breaks, names for each day, and reserving several spaces for addresses.

14.24.9 canvas

lv_canvas.h

Defines

LV_CANVAS_BUF_SIZE(w, h, bpp, stride)

Functions

lv_obj_t ***lv_canvas_create**(lv_obj_t *parent)

Create a canvas object

Parametri

parent -- pointer to an object, it will be the parent of the new canvas

Ritorna

pointer to the created canvas

void **lv_canvas_set_buffer**(lv_obj_t *obj, void *buf, int32_t w, int32_t h, *lv_color_format_t* cf)

Set a buffer for the canvas.

Use *lv_canvas_set_draw_buf()* instead if you need to set a buffer with alignment requirement.

Parametri

- **obj** -- pointer to a canvas object
- **buf** -- buffer where content of canvas will be. The required size is (lv_image_color_format_get_px_size(cf) * w) / 8 * h) It can be allocated with `lv_malloc()` or it can be statically allocated array (e.g. static *lv_color_t* buf[100*50]) or it can be an address in RAM or external SRAM
- **w** -- width of canvas
- **h** -- height of canvas
- **cf** -- color format. `LV_COLOR_FORMAT...`

void **lv_canvas_set_draw_buf**(lv_obj_t *obj, lv_draw_buf_t *draw_buf)

Set a draw buffer for the canvas. A draw buffer either can be allocated by `lv_draw_buf_create()` or defined statically by `LV_DRAW_BUF_DEFINE_STATIC`. When buffer start address and stride has alignment requirement, it's recommended to use `lv_draw_buf_create`.

Parametri

- **obj** -- pointer to a canvas object
- **draw_buf** -- pointer to a draw buffer

void **lv_canvas_set_px**(lv_obj_t *obj, int32_t x, int32_t y, *lv_color_t* color, lv_opa_t opa)

Set a pixel's color and opacity

> **Nota**
>
> The following color formats are supported LV_COLOR_FORMAT_I1/2/4/8, LV_COLOR_FORMAT_A8, LV_COLOR_FORMAT_RGB565, LV_COLOR_FORMAT_RGB888, LV_COLOR_FORMAT_XRGB8888, LV_COLOR_FORMAT_ARGB8888

Parametri

- **obj** -- pointer to a canvas
- **x** -- X coordinate of the pixel
- **y** -- Y coordinate of the pixel

- **color** -- the color
- **opa** -- the opacity

void **lv_canvas_set_palette**(lv_obj_t *obj, uint8_t index, *lv_color32_t* color)

Set the palette color of a canvas for index format. Valid only for `LV_COLOR_FORMAT_I1/2/4/8`

Parametri

- **obj** -- pointer to canvas object
- **index** -- the palette color to set:
  - for `LV_COLOR_FORMAT_I1`: 0..1
  - for `LV_COLOR_FORMAT_I2`: 0..3
  - for `LV_COLOR_FORMAT_I4`: 0..15
  - for `LV_COLOR_FORMAT_I8`: 0..255
- **color** -- the color to set

lv_draw_buf_t ***lv_canvas_get_draw_buf**(lv_obj_t *obj)

lv_color32_t **lv_canvas_get_px**(lv_obj_t *obj, int32_t x, int32_t y)

Get a pixel's color and opacity

Parametri

- **obj** -- pointer to a canvas
- **x** -- X coordinate of the pixel
- **y** -- Y coordinate of the pixel

Ritorna

ARGB8888 color of the pixel

lv_image_dsc_t ***lv_canvas_get_image**(lv_obj_t *canvas)

Get the image of the canvas as a pointer to an *lv_image_dsc_t* variable.

Parametri

canvas -- pointer to a canvas object

Ritorna

pointer to the image descriptor.

const void ***lv_canvas_get_buf**(lv_obj_t *canvas)

Return the pointer for the buffer. It's recommended to use this function instead of the buffer form the return value of *lv_canvas_get_image()* as is can be aligned

Parametri

canvas -- pointer to a canvas object

Ritorna

pointer to the buffer

void **lv_canvas_copy_buf**(lv_obj_t *obj, const *lv_area_t* *canvas_area, lv_draw_buf_t *dest_buf, const *lv_area_t* *dest_area)

Copy a buffer to the canvas

Parametri

- **obj** -- pointer to a canvas object
- **canvas_area** -- the area of the canvas to copy
- **dest_buf** -- pointer to a buffer to store the copied data
- **dest_area** -- the area of the destination buffer to copy to. If omitted NULL, copy to the whole `dest_buf`

void **lv_canvas_fill_bg**(lv_obj_t *obj, *lv_color_t* color, lv_opa_t opa)

Fill the canvas with color

Parametri

- **obj** -- pointer to a canvas
- **color** -- the background color
- **opa** -- the desired opacity

void **lv_canvas_init_layer**(lv_obj_t *canvas, lv_layer_t *layer)

Initialize a layer to use LVGL's generic draw functions (lv_draw_rect/label/...) on the canvas. Needs to be usd in pair with `lv_canvas_finish_layer`.

Parametri

- **canvas** -- pointer to a canvas
- **layer** -- pointer to a layer variable to initialize

void **lv_canvas_finish_layer**(lv_obj_t *canvas, lv_layer_t *layer)

Wait until all the drawings are finished on layer. Needs to be usd in pair with `lv_canvas_init_layer`.

Parametri

- **canvas** -- pointer to a canvas
- **layer** -- pointer to a layer to finalize

uint32_t **lv_canvas_buf_size**(int32_t w, int32_t h, uint8_t bpp, uint8_t stride)

Just a wrapper to `LV_CANVAS_BUF_SIZE` for bindings.

Variables

const lv_obj_class_t **lv_canvas_class**

lv_canvas_private.h

struct **_lv_canvas_t**

#include <lv_canvas_private.h> Canvas data

Public Members

lv_image_t **img**

lv_draw_buf_t ***draw_buf**

lv_draw_buf_t **static_buf**

14.24.10 chart

lv_chart.h

Defines

LV_CHART_POINT_NONE

Default value of points. Can be used to not draw a point

Enums

enum **lv_chart_type_t**

Chart types

Values:

enumerator **LV_CHART_TYPE_NONE**

Don't draw the series

enumerator **LV_CHART_TYPE_LINE**

Connect the points with lines

enumerator **LV_CHART_TYPE_BAR**

Draw columns

enumerator **LV_CHART_TYPE_SCATTER**

Draw points and lines in 2D (x,y coordinates)

enum **lv_chart_update_mode_t**

Chart update mode for `lv_chart_set_next`

Values:

enumerator **LV_CHART_UPDATE_MODE_SHIFT**

Shift old data to the left and add the new one the right

enumerator **LV_CHART_UPDATE_MODE_CIRCULAR**

Add the new data in a circular way

enum **lv_chart_axis_t**

Enumeration of the axis'

Values:

enumerator **LV_CHART_AXIS_PRIMARY_Y**

enumerator **LV_CHART_AXIS_SECONDARY_Y**

enumerator **LV_CHART_AXIS_PRIMARY_X**

enumerator **LV_CHART_AXIS_SECONDARY_X**

enumerator **LV_CHART_AXIS_LAST**

Functions

lv_obj_t ***lv_chart_create**(lv_obj_t *parent)

Create a chart object

Parametri

parent -- pointer to an object, it will be the parent of the new chart

Ritorna

pointer to the created chart

void **lv_chart_set_type**(lv_obj_t *obj, *lv_chart_type_t* type)

Set a new type for a chart

Parametri

- **obj** -- pointer to a chart object
- **type** -- new type of the chart (from '*lv_chart_type_t*' enum)

void **lv_chart_set_point_count**(lv_obj_t *obj, uint32_t cnt)

Set the number of points on a data line on a chart

Parametri

- **obj** -- pointer to a chart object
- **cnt** -- new number of points on the data lines

void **lv_chart_set_axis_range**(lv_obj_t *obj, *lv_chart_axis_t* axis, int32_t min, int32_t max)

Set the minimal and maximal y values on an axis

Parametri

- **obj** -- pointer to a chart object
- **axis** -- LV_CHART_AXIS_PRIMARY_Y or LV_CHART_AXIS_SECONDARY_Y
- **min** -- minimum value of the y axis
- **max** -- maximum value of the y axis

void **lv_chart_set_axis_min_value**(lv_obj_t *obj, *lv_chart_axis_t* axis, int32_t min)

Set the minimal values on an axis

Parametri

- **obj** -- pointer to a chart object
- **axis** -- LV_CHART_AXIS_PRIMARY_Y or LV_CHART_AXIS_SECONDARY_Y
- **min** -- minimal value of the y axis

void **lv_chart_set_axis_max_value**(lv_obj_t *obj, *lv_chart_axis_t* axis, int32_t max)

Set the maximal y values on an axis

Parametri

- **obj** -- pointer to a chart object
- **axis** -- LV_CHART_AXIS_PRIMARY_Y or LV_CHART_AXIS_SECONDARY_Y
- **max** -- maximum value of the y axis

void **lv_chart_set_update_mode**(lv_obj_t *obj, *lv_chart_update_mode_t* update_mode)

Set update mode of the chart object. Affects

Parametri

- **obj** -- pointer to a chart object
- **update_mode** -- the update mode

void **lv_chart_set_div_line_count**(lv_obj_t *obj, uint32_t hdiv, uint32_t vdiv)

Set the number of horizontal and vertical division lines

Parametri

- **obj** -- pointer to a chart object
- **hdiv** -- number of horizontal division lines
- **vdiv** -- number of vertical division lines

void **lv_chart_set_hor_div_line_count**(lv_obj_t *obj, uint32_t cnt)

Set the number of horizontal division lines

Parametri

- **obj** -- pointer to a chart object
- **cnt** -- number of horizontal division lines

void **lv_chart_set_ver_div_line_count**(lv_obj_t *obj, uint32_t cnt)

Set the number of vertical division lines

Parametri

- **obj** -- pointer to a chart object
- **cnt** -- number of vertical division lines

lv_chart_type_t **lv_chart_get_type**(const lv_obj_t *obj)

Get the type of a chart

Parametri

obj -- pointer to chart object

Ritorna
type of the chart (from 'lv_chart_t' enum)

uint32_t **lv_chart_get_point_count**(const lv_obj_t *obj)

Get the data point number per data line on chart

Parametri
obj -- pointer to chart object

Ritorna
point number on each data line

uint32_t **lv_chart_get_x_start_point**(const lv_obj_t *obj, lv_chart_series_t *ser)

Get the current index of the x-axis start point in the data array

Parametri
- **obj** -- pointer to a chart object
- **ser** -- pointer to a data series on 'chart'

Ritorna
the index of the current x start point in the data array

void **lv_chart_get_point_pos_by_id**(lv_obj_t *obj, lv_chart_series_t *ser, uint32_t id, *lv_point_t* *p_out)

Get the position of a point to the chart.

Parametri
- **obj** -- pointer to a chart object
- **ser** -- pointer to series
- **id** -- the index.
- **p_out** -- store the result position here

void **lv_chart_refresh**(lv_obj_t *obj)

Refresh a chart if its data line has changed

Parametri
obj -- pointer to chart object

lv_chart_series_t ***lv_chart_add_series**(lv_obj_t *obj, *lv_color_t* color, *lv_chart_axis_t* axis)

Allocate and add a data series to the chart

Parametri
- **obj** -- pointer to a chart object
- **color** -- color of the data series
- **axis** -- the y axis to which the series should be attached (::LV_CHART_AXIS_PRIMARY_Y or ::LV_CHART_AXIS_SECONDARY_Y)

Ritorna
pointer to the allocated data series or NULL on failure

void **lv_chart_remove_series**(lv_obj_t *obj, lv_chart_series_t *series)

Deallocate and remove a data series from a chart

Parametri
- **obj** -- pointer to a chart object
- **series** -- pointer to a data series on 'chart'

void **lv_chart_hide_series**(lv_obj_t *chart, lv_chart_series_t *series, bool hide)

Hide/Unhide a single series of a chart.

Parametri
- **chart** -- pointer to a chart object.
- **series** -- pointer to a series object
- **hide** -- true: hide the series

void **lv_chart_set_series_color**(lv_obj_t *chart, lv_chart_series_t *series, *lv_color_t* color)

Change the color of a series

Parametri
- **chart** -- pointer to a chart object.
- **series** -- pointer to a series object

- **color** -- the new color of the series

lv_color_t **lv_chart_get_series_color**(lv_obj_t *chart, const lv_chart_series_t *series)

Get the color of a series

Parametri

- **chart** -- pointer to a chart object.
- **series** -- pointer to a series object

Ritorna

the color of the series

void **lv_chart_set_x_start_point**(lv_obj_t *obj, lv_chart_series_t *ser, uint32_t id)

Set the index of the x-axis start point in the data array. This point will be considers the first (left) point and the other points will be drawn after it.

Parametri

- **obj** -- pointer to a chart object
- **ser** -- pointer to a data series on 'chart'
- **id** -- the index of the x point in the data array

lv_chart_series_t ***lv_chart_get_series_next**(const lv_obj_t *chart, const lv_chart_series_t *ser)

Get the next series.

Parametri

- **chart** -- pointer to a chart
- **ser** -- the previous series or NULL to get the first

Ritorna

the next series or NULL if there is no more.

lv_chart_cursor_t ***lv_chart_add_cursor**(lv_obj_t *obj, *lv_color_t* color, *lv_dir_t* dir)

Add a cursor with a given color

Parametri

- **obj** -- pointer to chart object
- **color** -- color of the cursor
- **dir** -- direction of the cursor. `LV_DIR_RIGHT/LEFT/TOP/DOWN/HOR/VER/ALL`. OR-ed values are possible

Ritorna

pointer to the created cursor

void **lv_chart_set_cursor_pos**(lv_obj_t *chart, lv_chart_cursor_t *cursor, *lv_point_t* *pos)

Set the coordinate of the cursor with respect to the paddings

Parametri

- **chart** -- pointer to a chart object
- **cursor** -- pointer to the cursor
- **pos** -- the new coordinate of cursor relative to the chart

void **lv_chart_set_cursor_pos_x**(lv_obj_t *chart, lv_chart_cursor_t *cursor, int32_t x)

Set the X coordinate of the cursor with respect to the paddings

Parametri

- **chart** -- pointer to a chart object
- **cursor** -- pointer to the cursor
- **x** -- the new X coordinate of cursor relative to the chart

void **lv_chart_set_cursor_pos_y**(lv_obj_t *chart, lv_chart_cursor_t *cursor, int32_t y)

Set the coordinate of the cursor with respect to the paddings

Parametri

- **chart** -- pointer to a chart object
- **cursor** -- pointer to the cursor
- **y** -- the new Y coordinate of cursor relative to the chart

void **lv_chart_set_cursor_point**(lv_obj_t *chart, lv_chart_cursor_t *cursor, lv_chart_series_t *ser, uint32_t point_id)

Stick the cursor to a point

Parametri
- **chart** -- pointer to a chart object
- **cursor** -- pointer to the cursor
- **ser** -- pointer to a series
- **point_id** -- the point's index or `LV_CHART_POINT_NONE` to not assign to any points.

lv_point_t **lv_chart_get_cursor_point**(lv_obj_t *chart, lv_chart_cursor_t *cursor)

Get the coordinate of the cursor with respect to the paddings

Parametri
- **chart** -- pointer to a chart object
- **cursor** -- pointer to cursor

Ritorna
coordinate of the cursor as *lv_point_t*

void **lv_chart_set_all_values**(lv_obj_t *obj, lv_chart_series_t *ser, int32_t value)

Initialize all data points of a series with a value

Parametri
- **obj** -- pointer to chart object
- **ser** -- pointer to a data series on 'chart'
- **value** -- the new value for all points. `LV_CHART_POINT_NONE` can be used to hide the points.

void **lv_chart_set_next_value**(lv_obj_t *obj, lv_chart_series_t *ser, int32_t value)

Set the next point's Y value according to the update mode policy.

Parametri
- **obj** -- pointer to chart object
- **ser** -- pointer to a data series on 'chart'
- **value** -- the new value of the next data

void **lv_chart_set_next_value2**(lv_obj_t *obj, lv_chart_series_t *ser, int32_t x_value, int32_t y_value)

Set the next point's X and Y value according to the update mode policy.

Parametri
- **obj** -- pointer to chart object
- **ser** -- pointer to a data series on 'chart'
- **x_value** -- the new X value of the next data
- **y_value** -- the new Y value of the next data

void **lv_chart_set_series_values**(lv_obj_t *obj, lv_chart_series_t *ser, const int32_t values[], size_t values_cnt)

Same as `lv_chart_set_next_value` but set the values from an array

Parametri
- **obj** -- pointer to chart object
- **ser** -- pointer to a data series on 'chart'
- **values** -- the new values to set
- **values_cnt** -- number of items in `values`

void **lv_chart_set_series_values2**(lv_obj_t *obj, lv_chart_series_t *ser, const int32_t x_values[], const int32_t y_values[], size_t values_cnt)

Same as `lv_chart_set_next_value2` but set the values from an array

Parametri
- **obj** -- pointer to chart object
- **ser** -- pointer to a data series on 'chart'
- **x_values** -- the new values to set on the X axis
- **y_values** -- the new values to set o nthe Y axis
- **values_cnt** -- number of items in `x_values` and `y_values`

void **lv_chart_set_series_value_by_id**(lv_obj_t *obj, lv_chart_series_t *ser, uint32_t id, int32_t value)

Set an individual point's y value of a chart's series directly based on its index

Parametri

- **obj** -- pointer to a chart object
- **ser** -- pointer to a data series on 'chart'
- **id** -- the index of the x point in the array
- **value** -- value to assign to array point

void **lv_chart_set_series_value_by_id2**(lv_obj_t *obj, lv_chart_series_t *ser, uint32_t id, int32_t x_value, int32_t y_value)

Set an individual point's x and y value of a chart's series directly based on its index Can be used only with `LV_CHART_TYPE_SCATTER`.

Parametri

- **obj** -- pointer to chart object
- **ser** -- pointer to a data series on 'chart'
- **id** -- the index of the x point in the array
- **x_value** -- the new X value of the next data
- **y_value** -- the new Y value of the next data

void **lv_chart_set_series_ext_y_array**(lv_obj_t *obj, lv_chart_series_t *ser, int32_t array[])

Set an external array for the y data points to use for the chart NOTE: It is the users responsibility to make sure the `point_cnt` matches the external array size.

Parametri

- **obj** -- pointer to a chart object
- **ser** -- pointer to a data series on 'chart'
- **array** -- external array of points for chart

void **lv_chart_set_series_ext_x_array**(lv_obj_t *obj, lv_chart_series_t *ser, int32_t array[])

Set an external array for the x data points to use for the chart NOTE: It is the users responsibility to make sure the `point_cnt` matches the external array size.

Parametri

- **obj** -- pointer to a chart object
- **ser** -- pointer to a data series on 'chart'
- **array** -- external array of points for chart

int32_t ***lv_chart_get_series_y_array**(const lv_obj_t *obj, lv_chart_series_t *ser)

Get the array of y values of a series

Parametri

- **obj** -- pointer to a chart object
- **ser** -- pointer to a data series on 'chart'

Ritorna

the array of values with 'point_count' elements

int32_t ***lv_chart_get_series_x_array**(const lv_obj_t *obj, lv_chart_series_t *ser)

Get the array of x values of a series

Parametri

- **obj** -- pointer to a chart object
- **ser** -- pointer to a data series on 'chart'

Ritorna

the array of values with 'point_count' elements

uint32_t **lv_chart_get_pressed_point**(const lv_obj_t *obj)

Get the index of the currently pressed point. It's the same for every series.

Parametri

obj -- pointer to a chart object

Ritorna

the index of the point [0 .. point count] or LV_CHART_POINT_ID_NONE if no point is being pressed

int32_t **lv_chart_get_first_point_center_offset**(lv_obj_t *obj)

Get the overall offset from the chart's side to the center of the first point. In case of a bar chart it will be the center of the first column group

Parametri
obj -- pointer to a chart object
Ritorna
the offset of the center

Variables

const lv_obj_class_t **lv_chart_class**

lv_chart_private.h

struct **_lv_chart_series_t**

#include <lv_chart_private.h> Descriptor a chart series

Public Members

int32_t ***x_points**

int32_t ***y_points**

lv_color_t **color**

uint32_t **start_point**

uint32_t **hidden**

uint32_t **x_ext_buf_assigned**

uint32_t **y_ext_buf_assigned**

uint32_t **x_axis_sec**

uint32_t **y_axis_sec**

struct **_lv_chart_cursor_t**

Public Members

lv_point_t **pos**

int32_t **point_id**

lv_color_t **color**

lv_chart_series_t ***ser**

lv_dir_t **dir**

uint32_t **pos_set**

1: pos is set; 0: point_id is set

struct **_lv_chart_t**

Public Members

lv_obj_t **obj**

lv_ll_t **series_ll**

Linked list for series (stores lv_chart_series_t)

lv_ll_t **cursor_ll**

Linked list for cursors (stores lv_chart_cursor_t)

int32_t **ymin**[2]

int32_t **ymax**[2]

int32_t **xmin**[2]

int32_t **xmax**[2]

int32_t **pressed_point_id**

uint32_t **hdiv_cnt**

Number of horizontal division lines

uint32_t **vdiv_cnt**

Number of vertical division lines

uint32_t **point_cnt**

Number of points in all series

lv_chart_type_t **type**

Chart type

lv_chart_update_mode_t **update_mode**

14.24.11 checkbox

lv_checkbox.h

Functions

lv_obj_t ***lv_checkbox_create**(lv_obj_t *parent)

Create a check box object

Parametri

parent -- pointer to an object, it will be the parent of the new button

Ritorna
pointer to the created check box

void **lv_checkbox_set_text**(lv_obj_t *obj, const char *txt)

Set the text of a check box. `txt` will be copied and may be deallocated after this function returns.

Parametri
- **obj** -- pointer to a check box
- **txt** -- the text of the check box. NULL to refresh with the current text.

void **lv_checkbox_set_text_static**(lv_obj_t *obj, const char *txt)

Set the text of a check box. `txt` must not be deallocated during the life of this checkbox.

Parametri
- **obj** -- pointer to a check box
- **txt** -- the text of the check box.

const char ***lv_checkbox_get_text**(const lv_obj_t *obj)

Get the text of a check box

Parametri
obj -- pointer to check box object

Ritorna
pointer to the text of the check box

Variables

const lv_obj_class_t **lv_checkbox_class**

lv_checkbox_private.h

struct **_lv_checkbox_t**

Public Members

lv_obj_t **obj**

char ***txt**

uint32_t **static_txt**

14.24.12 dropdown

lv_dropdown.h

Defines

LV_DROPDOWN_POS_LAST

Enums

enum **_lv_property_dropdown_id_t**

Values:

enumerator **LV_PROPERTY_DROPDOWN_TEXT**

enumerator **LV_PROPERTY_DROPDOWN_OPTIONS**

enumerator **LV_PROPERTY_DROPDOWN_OPTION_COUNT**

enumerator **LV_PROPERTY_DROPDOWN_SELECTED**

enumerator **LV_PROPERTY_DROPDOWN_DIR**

enumerator **LV_PROPERTY_DROPDOWN_SYMBOL**

enumerator **LV_PROPERTY_DROPDOWN_SELECTED_HIGHLIGHT**

enumerator **LV_PROPERTY_DROPDOWN_LIST**

enumerator **LV_PROPERTY_DROPDOWN_IS_OPEN**

enumerator **LV_PROPERTY_DROPDOWN_END**

Functions

lv_obj_t ***lv_dropdown_create**(lv_obj_t *parent)

Create a drop-down list object

Parametri

parent -- pointer to an object, it will be the parent of the new drop-down list

Ritorna

pointer to the created drop-down list

void **lv_dropdown_set_text**(lv_obj_t *obj, const char *txt)

Set text of the drop-down list's button. If set to `NULL` the selected option's text will be displayed on the button. If set to a specific text then that text will be shown regardless of the selected option.

Parametri

- **obj** -- pointer to a drop-down list object
- **txt** -- the text as a string (Only its pointer is saved)

void **lv_dropdown_set_options**(lv_obj_t *obj, const char *options)

Set the options in a drop-down list from a string. The options will be copied and saved in the object so the `options` can be destroyed after calling this function

Parametri

- **obj** -- pointer to drop-down list object
- **options** -- a string with '

  ' separated options. E.g. "One\nTwo\nThree"

void **lv_dropdown_set_options_static**(lv_obj_t *obj, const char *options)

Set the options in a drop-down list from a static string (global, static or dynamically allocated). Only the pointer of the option string will be saved.

Parametri

- **obj** -- pointer to drop-down list object
- **options** -- a static string with '

  ' separated options. E.g. "One\nTwo\nThree"

void **lv_dropdown_add_option**(lv_obj_t *obj, const char *option, uint32_t pos)

Add an options to a drop-down list from a string. Only works for non-static options.

Parametri

- **obj** -- pointer to drop-down list object
- **option** -- a string without '

  '. E.g. "Four"
- **pos** -- the insert position, indexed from 0, LV_DROPDOWN_POS_LAST = end of string

void **lv_dropdown_clear_options**(lv_obj_t *obj)

Clear all options in a drop-down list. Works with both static and dynamic options.

Parametri

obj -- pointer to drop-down list object

void **lv_dropdown_set_selected**(lv_obj_t *obj, uint32_t sel_opt)

Set the selected option

Parametri

- **obj** -- pointer to drop-down list object
- **sel_opt** -- id of the selected option (0 ... number of option - 1);

void **lv_dropdown_set_dir**(lv_obj_t *obj, *lv_dir_t* dir)

Set the direction of the a drop-down list

Parametri

- **obj** -- pointer to a drop-down list object
- **dir** -- LV_DIR_LEFT/RIGHT/TOP/BOTTOM

void **lv_dropdown_set_symbol**(lv_obj_t *obj, const void *symbol)

Set an arrow or other symbol to display when on drop-down list's button. Typically a down caret or arrow.

> **Nota**
>
> angle and zoom transformation can be applied if the symbol is an image. E.g. when drop down is checked (opened) rotate the symbol by 180 degree

Parametri

- **obj** -- pointer to drop-down list object
- **symbol** -- a text like `LV_SYMBOL_DOWN`, an image (pointer or path) or NULL to not draw symbol icon

void **lv_dropdown_set_selected_highlight**(lv_obj_t *obj, bool en)

Set whether the selected option in the list should be highlighted or not

Parametri

- **obj** -- pointer to drop-down list object
- **en** -- true: highlight enabled; false: disabled

lv_obj_t ***lv_dropdown_get_list**(lv_obj_t *obj)

Get the list of a drop-down to allow styling or other modifications

Parametri

obj -- pointer to a drop-down list object

Ritorna

pointer to the list of the drop-down

const char ***lv_dropdown_get_text**(lv_obj_t *obj)

Get text of the drop-down list's button.

Parametri

obj -- pointer to a drop-down list object

Ritorna

the text as string, `NULL` if no text

const char ***lv_dropdown_get_options**(const lv_obj_t *obj)

Get the options of a drop-down list

Parametri
obj -- pointer to drop-down list object
Ritorna
the options separated by '

'-s (E.g. "Option1\nOption2\nOption3")

uint32_t **lv_dropdown_get_selected**(const lv_obj_t *obj)

Get the index of the selected option
Parametri
obj -- pointer to drop-down list object
Ritorna
index of the selected option (0 ... number of option - 1);

uint32_t **lv_dropdown_get_option_count**(const lv_obj_t *obj)

Get the total number of options
Parametri
obj -- pointer to drop-down list object
Ritorna
the total number of options in the list

void **lv_dropdown_get_selected_str**(const lv_obj_t *obj, char *buf, uint32_t buf_size)

Get the current selected option as a string
Parametri
- **obj** -- pointer to drop-down object
- **buf** -- pointer to an array to store the string
- **buf_size** -- size of `buf` in bytes. 0: to ignore it.

int32_t **lv_dropdown_get_option_index**(lv_obj_t *obj, const char *option)

Get the index of an option.
Parametri
- **obj** -- pointer to drop-down object
- **option** -- an option as string

Ritorna
index of `option` in the list of all options. -1 if not found.

const char ***lv_dropdown_get_symbol**(lv_obj_t *obj)

Get the symbol on the drop-down list. Typically a down caret or arrow.
Parametri
obj -- pointer to drop-down list object
Ritorna
the symbol or NULL if not enabled

bool **lv_dropdown_get_selected_highlight**(lv_obj_t *obj)

Get whether the selected option in the list should be highlighted or not
Parametri
obj -- pointer to drop-down list object
Ritorna
true: highlight enabled; false: disabled

lv_dir_t **lv_dropdown_get_dir**(const lv_obj_t *obj)

Get the direction of the drop-down list
Parametri
obj -- pointer to a drop-down list object
Ritorna
LV_DIR_LEF/RIGHT/TOP/BOTTOM

void **lv_dropdown_open**(lv_obj_t *dropdown_obj)

Open the drop.down list

Parametri
dropdown_obj -- pointer to drop-down list object

void **lv_dropdown_close**(lv_obj_t *obj)

Close (Collapse) the drop-down list

Parametri
obj -- pointer to drop-down list object

bool **lv_dropdown_is_open**(lv_obj_t *obj)

Tells whether the list is opened or not

Parametri
obj -- pointer to a drop-down list object

Ritorna
true if the list os opened

Variables

const lv_obj_class_t **lv_dropdown_class**

const lv_obj_class_t **lv_dropdownlist_class**

lv_dropdown_private.h

struct **_lv_dropdown_t**

Public Members

lv_obj_t **obj**

lv_obj_t ***list**

The dropped down list

const char ***text**

Text to display on the dropdown's button

const void ***symbol**

Arrow or other icon when the drop-down list is closed

char ***options**

Options in a '

' separated list

uint32_t **option_cnt**

Number of options

uint32_t **sel_opt_id**

Index of the currently selected option

uint32_t **sel_opt_id_orig**

Store the original index on focus

uint32_t **pr_opt_id**

Index of the currently pressed option

uint8_t **dir**

Direction in which the list should open

uint8_t **static_txt**

1: Only a pointer is saved in `options`

uint8_t **selected_highlight**

1: Make the selected option highlighted in the list

struct **_lv_dropdown_list_t**

Public Members

lv_obj_t **obj**

lv_obj_t ***dropdown**

14.24.13 image

lv_image.h

Defines

LV_IMAGE_DECLARE(var_name)

Use this macro to declare an image in a C file

Enums

enum **lv_image_align_t**

Image size mode, when image size and object size is different

Values:

enumerator **LV_IMAGE_ALIGN_DEFAULT**

enumerator **LV_IMAGE_ALIGN_TOP_LEFT**

enumerator **LV_IMAGE_ALIGN_TOP_MID**

enumerator **LV_IMAGE_ALIGN_TOP_RIGHT**

enumerator **LV_IMAGE_ALIGN_BOTTOM_LEFT**

enumerator **LV_IMAGE_ALIGN_BOTTOM_MID**

enumerator **LV_IMAGE_ALIGN_BOTTOM_RIGHT**

enumerator **LV_IMAGE_ALIGN_LEFT_MID**

enumerator **LV_IMAGE_ALIGN_RIGHT_MID**

enumerator **LV_IMAGE_ALIGN_CENTER**

enumerator **LV_IMAGE_ALIGN_AUTO_TRANSFORM**

enumerator **LV_IMAGE_ALIGN_STRETCH**

enumerator **LV_IMAGE_ALIGN_TILE**

enumerator **LV_IMAGE_ALIGN_CONTAIN**

enumerator **LV_IMAGE_ALIGN_COVER**

enum **_lv_property_image_id_t**

Values:

enumerator **LV_PROPERTY_IMAGE_SRC**

enumerator **LV_PROPERTY_IMAGE_OFFSET_X**

enumerator **LV_PROPERTY_IMAGE_OFFSET_Y**

enumerator **LV_PROPERTY_IMAGE_ROTATION**

enumerator **LV_PROPERTY_IMAGE_PIVOT**

enumerator **LV_PROPERTY_IMAGE_SCALE**

enumerator **LV_PROPERTY_IMAGE_SCALE_X**

enumerator **LV_PROPERTY_IMAGE_SCALE_Y**

enumerator **LV_PROPERTY_IMAGE_BLEND_MODE**

enumerator **LV_PROPERTY_IMAGE_ANTIALIAS**

enumerator **LV_PROPERTY_IMAGE_INNER_ALIGN**

enumerator **LV_PROPERTY_IMAGE_END**

Functions

lv_obj_t *`lv_image_create`(lv_obj_t *parent)

Create an image object

Parametri

parent -- pointer to an object, it will be the parent of the new image

Ritorna

pointer to the created image

void `lv_image_set_src`(lv_obj_t *obj, const void *src)

Set the image data to display on the object

Parametri

- **obj** -- pointer to an image object
- **src** -- 1) pointer to an *lv_image_dsc_t* descriptor (converted by LVGL's image converter) (e.g. &my_img) or 2) path to an image file (e.g. "S:/dir/img.bin")or 3) a SYMBOL (e.g. LV_SYMBOL_OK)

void `lv_image_set_offset_x`(lv_obj_t *obj, int32_t x)

Set an offset for the source of an image so the image will be displayed from the new origin.

Parametri

- **obj** -- pointer to an image
- **x** -- the new offset along x axis.

void `lv_image_set_offset_y`(lv_obj_t *obj, int32_t y)

Set an offset for the source of an image. so the image will be displayed from the new origin.

Parametri

- **obj** -- pointer to an image
- **y** -- the new offset along y axis.

void `lv_image_set_rotation`(lv_obj_t *obj, int32_t angle)

Set the rotation angle of the image. The image will be rotated around the set pivot set by *`lv_image_set_pivot()`* Note that indexed and alpha only images can't be transformed.

> **Nota**
>
> if image_align is `LV_IMAGE_ALIGN_STRETCH` or `LV_IMAGE_ALIGN_FIT` rotation will be set to 0 automatically.

Parametri

- **obj** -- pointer to an image object
- **angle** -- rotation in degree with 0.1 degree resolution (0..3600: clock wise)

void `lv_image_set_pivot`(lv_obj_t *obj, int32_t x, int32_t y)

Set the rotation center of the image. The image will be rotated around this point. x, y can be set with value of LV_PCT, lv_image_get_pivot will return the true pixel coordinate of pivot in this case.

Parametri

- **obj** -- pointer to an image object
- **x** -- rotation center x of the image
- **y** -- rotation center y of the image

void `lv_image_set_pivot_x`(lv_obj_t *obj, int32_t x)

Set the rotation horizontal center of the image.

Parametri

- **obj** -- pointer to an image object
- **x** -- rotation center x of the image, or *lv_pct()*

void `lv_image_set_pivot_y`(lv_obj_t *obj, int32_t y)

Set the rotation vertical center of the image.

Parametri

- **obj** -- pointer to an image object

- **y** -- rotation center y of the image, or *lv_pct()*

void **lv_image_set_scale**(lv_obj_t *obj, uint32_t zoom)

Set the zoom factor of the image. Note that indexed and alpha only images can't be transformed.

Parametri

- **obj** -- pointer to an image object
- **zoom** -- the zoom factor. Example values:
  - 256 or LV_SCALE_NONE: no zoom
  - <256: scale down
  - >256: scale up
  - 128: half size
  - 512: double size

void **lv_image_set_scale_x**(lv_obj_t *obj, uint32_t zoom)

Set the horizontal zoom factor of the image. Note that indexed and alpha only images can't be transformed.

Parametri

- **obj** -- pointer to an image object
- **zoom** -- the zoom factor. Example values:
  - 256 or LV_SCALE_NONE: no zoom
  - <256: scale down
  - >256: scale up
  - 128: half size
  - 512: double size

void **lv_image_set_scale_y**(lv_obj_t *obj, uint32_t zoom)

Set the vertical zoom factor of the image. Note that indexed and alpha only images can't be transformed.

Parametri

- **obj** -- pointer to an image object
- **zoom** -- the zoom factor. Example values:
  - 256 or LV_SCALE_NONE: no zoom
  - <256: scale down
  - >256: scale up
  - 128: half size
  - 512: double size

void **lv_image_set_blend_mode**(lv_obj_t *obj, *lv_blend_mode_t* blend_mode)

Set the blend mode of an image.

Parametri

- **obj** -- pointer to an image object
- **blend_mode** -- the new blend mode

void **lv_image_set_antialias**(lv_obj_t *obj, bool antialias)

Enable/disable anti-aliasing for the transformations (rotate, zoom) or not. The quality is better with anti-aliasing looks better but slower.

Parametri

- **obj** -- pointer to an image object
- **antialias** -- true: anti-aliased; false: not anti-aliased

void **lv_image_set_inner_align**(lv_obj_t *obj, *lv_image_align_t* align)

Set the image object size mode.

> **Nota**
>
> if image_align is `LV_IMAGE_ALIGN_STRETCH` or `LV_IMAGE_ALIGN_FIT` rotation, scale and pivot will be overwritten and controlled internally.

Parametri
- **obj** -- pointer to an image object
- **align** -- the new align mode.

void **lv_image_set_bitmap_map_src**(lv_obj_t *obj, const *lv_image_dsc_t* *src)

Set an A8 bitmap mask for the image.

Parametri
- **obj** -- pointer to an image object
- **src** -- an *lv_image_dsc_t* bitmap mask source.

const void ***lv_image_get_src**(lv_obj_t *obj)

Get the source of the image

Parametri
obj -- pointer to an image object

Ritorna
the image source (symbol, file name or ::lv-img_dsc_t for C arrays)

int32_t **lv_image_get_offset_x**(lv_obj_t *obj)

Get the offset's x attribute of the image object.

Parametri
obj -- pointer to an image

Ritorna
offset X value.

int32_t **lv_image_get_offset_y**(lv_obj_t *obj)

Get the offset's y attribute of the image object.

Parametri
obj -- pointer to an image

Ritorna
offset Y value.

int32_t **lv_image_get_rotation**(lv_obj_t *obj)

Get the rotation of the image.

> **Nota**
>
> if image_align is `LV_IMAGE_ALIGN_STRETCH` or `LV_IMAGE_ALIGN_FIT` rotation will be set to 0 automatically.

Parametri
obj -- pointer to an image object

Ritorna
rotation in 0.1 degrees (0..3600)

void **lv_image_get_pivot**(lv_obj_t *obj, *lv_point_t* *pivot)

Get the pivot (rotation center) of the image. If pivot is set with LV_PCT, convert it to px before return.

Parametri
- **obj** -- pointer to an image object
- **pivot** -- store the rotation center here

int32_t **lv_image_get_scale**(lv_obj_t *obj)

Get the zoom factor of the image.

Parametri
obj -- pointer to an image object

Ritorna
zoom factor (256: no zoom)

int32_t **lv_image_get_scale_x**(lv_obj_t *obj)

Get the horizontal zoom factor of the image.

Parametri
obj -- pointer to an image object

Ritorna
zoom factor (256: no zoom)

int32_t **lv_image_get_scale_y**(lv_obj_t *obj)

Get the vertical zoom factor of the image.

Parametri
obj -- pointer to an image object

Ritorna
zoom factor (256: no zoom)

int32_t **lv_image_get_src_width**(lv_obj_t *obj)

Get the width of an image before any transformations.

Parametri
obj -- Pointer to an image object.

Ritorna
The width of the image.

int32_t **lv_image_get_src_height**(lv_obj_t *obj)

Get the height of an image before any transformations.

Parametri
obj -- Pointer to an image object.

Ritorna
The height of the image.

int32_t **lv_image_get_transformed_width**(lv_obj_t *obj)

Get the transformed width of an image object.

Parametri
obj -- Pointer to an image object.

Ritorna
The transformed width of the image.

int32_t **lv_image_get_transformed_height**(lv_obj_t *obj)

Get the transformed height of an image object.

Parametri
obj -- Pointer to an image object.

Ritorna
The transformed height of the image.

lv_blend_mode_t **lv_image_get_blend_mode**(lv_obj_t *obj)

Get the current blend mode of the image

Parametri
obj -- pointer to an image object

Ritorna
the current blend mode

bool **lv_image_get_antialias**(lv_obj_t *obj)

Get whether the transformations (rotate, zoom) are anti-aliased or not

Parametri
obj -- pointer to an image object

Ritorna
true: anti-aliased; false: not anti-aliased

lv_image_align_t **lv_image_get_inner_align**(lv_obj_t *obj)

Get the size mode of the image

Parametri
obj -- pointer to an image object
Ritorna
element of *lv_image_align_t*

const *lv_image_dsc_t* ***lv_image_get_bitmap_map_src**(lv_obj_t *obj)

Get the bitmap mask source.
Parametri
obj -- pointer to an image object
Ritorna
an *lv_image_dsc_t* bitmap mask source.

Variables

const lv_obj_class_t **lv_image_class**

lv_image_private.h

struct **_lv_image_t**

#include <lv_image_private.h> Data of image

Public Members

lv_obj_t **obj**

const void ***src**

Image source: Pointer to an array or a file or a symbol

const *lv_image_dsc_t* ***bitmap_mask_src**

Pointer to an A8 bitmap mask

lv_point_t **offset**

int32_t **w**

Width of the image (Handled by the library)

int32_t **h**

Height of the image (Handled by the library)

uint32_t **rotation**

Rotation angle of the image

uint32_t **scale_x**

256 means no zoom, 512 double size, 128 half size

uint32_t **scale_y**

256 means no zoom, 512 double size, 128 half size

lv_point_t **pivot**

Rotation center of the image

uint32_t **src_type**

See: *lv_image_src_t*

uint32_t **cf**

Color format from `lv_color_format_t`

uint32_t **antialias**

Apply anti-aliasing in transformations (rotate, zoom)

uint32_t **align**

Image size mode when image size and object size is different. See *lv_image_align_t*

uint32_t **blend_mode**

Element of `lv_blend_mode_t`

14.24.14 imagebutton

lv_imagebutton.h

Enums

enum **lv_imagebutton_state_t**

Values:

enumerator **LV_IMAGEBUTTON_STATE_RELEASED**

enumerator **LV_IMAGEBUTTON_STATE_PRESSED**

enumerator **LV_IMAGEBUTTON_STATE_DISABLED**

enumerator **LV_IMAGEBUTTON_STATE_CHECKED_RELEASED**

enumerator **LV_IMAGEBUTTON_STATE_CHECKED_PRESSED**

enumerator **LV_IMAGEBUTTON_STATE_CHECKED_DISABLED**

enumerator **LV_IMAGEBUTTON_STATE_NUM**

Functions

lv_obj_t ***lv_imagebutton_create**(lv_obj_t *parent)

Create an image button object

Parametri

parent -- pointer to an object, it will be the parent of the new image button

Ritorna

pointer to the created image button

void **lv_imagebutton_set_src**(lv_obj_t *imagebutton, *lv_imagebutton_state_t* state, const void *src_left, const void *src_mid, const void *src_right)

Set images for a state of the image button

Parametri

- **imagebutton** -- pointer to an image button object
- **state** -- for which state set the new image
- **src_left** -- pointer to an image source for the left side of the button (a C array or path to a file)
- **src_mid** -- pointer to an image source for the middle of the button (ideally 1px wide) (a C array or path to a file)
- **src_right** -- pointer to an image source for the right side of the button (a C array or path to a file)

void **lv_imagebutton_set_state**(lv_obj_t *imagebutton, *lv_imagebutton_state_t* state)

Use this function instead of `lv_obj_add/remove_state` to set a state manually

Parametri

- **imagebutton** -- pointer to an image button object
- **state** -- the new state

const void ***lv_imagebutton_get_src_left**(lv_obj_t *imagebutton, *lv_imagebutton_state_t* state)

Get the left image in a given state

Parametri

- **imagebutton** -- pointer to an image button object
- **state** -- the state where to get the image (from `lv_button_state_t`)`

Ritorna

pointer to the left image source (a C array or path to a file)

const void ***lv_imagebutton_get_src_middle**(lv_obj_t *imagebutton, *lv_imagebutton_state_t* state)

Get the middle image in a given state

Parametri

- **imagebutton** -- pointer to an image button object
- **state** -- the state where to get the image (from `lv_button_state_t`)`

Ritorna

pointer to the middle image source (a C array or path to a file)

const void ***lv_imagebutton_get_src_right**(lv_obj_t *imagebutton, *lv_imagebutton_state_t* state)

Get the right image in a given state

Parametri

- **imagebutton** -- pointer to an image button object
- **state** -- the state where to get the image (from `lv_button_state_t`)`

Ritorna

pointer to the left image source (a C array or path to a file)

Variables

const lv_obj_class_t **lv_imagebutton_class**

lv_imagebutton_private.h

struct **_lv_imagebutton_src_info_t**

Public Members

const void ***img_src**

lv_image_header_t **header**

struct **_lv_imagebutton_t**

#include <lv_imagebutton_private.h> Data of image button

Public Members

lv_obj_t **obj**

lv_imagebutton_src_info_t **src_mid**[*LV_IMAGEBUTTON_STATE_NUM*]

Store center images to each state

lv_imagebutton_src_info_t **src_left**[*LV_IMAGEBUTTON_STATE_NUM*]

Store left side images to each state

lv_imagebutton_src_info_t **src_right**[*LV_IMAGEBUTTON_STATE_NUM*]

Store right side images to each state

14.24.15 keyboard

lv_keyboard.h

Defines

LV_KEYBOARD_CTRL_BUTTON_FLAGS

Enums

enum **lv_keyboard_mode_t**

Current keyboard mode.

Values:

enumerator **LV_KEYBOARD_MODE_TEXT_LOWER**

enumerator **LV_KEYBOARD_MODE_TEXT_UPPER**

enumerator **LV_KEYBOARD_MODE_SPECIAL**

enumerator **LV_KEYBOARD_MODE_NUMBER**

enumerator **LV_KEYBOARD_MODE_USER_1**

enumerator **LV_KEYBOARD_MODE_USER_2**

enumerator **LV_KEYBOARD_MODE_USER_3**

enumerator **LV_KEYBOARD_MODE_USER_4**

enumerator **LV_KEYBOARD_MODE_TEXT_ARABIC**

enum **_lv_property_keyboard_id_t**

Values:

enumerator **LV_PROPERTY_KEYBOARD_TEXTAREA**

enumerator **LV_PROPERTY_KEYBOARD_MODE**

enumerator **LV_PROPERTY_KEYBOARD_POPOVERS**

enumerator **LV_PROPERTY_KEYBOARD_SELECTED_BUTTON**

enumerator **LV_PROPERTY_KEYBOARD_END**

Functions

lv_obj_t ***lv_keyboard_create**(lv_obj_t *parent)

Create a Keyboard object

Parametri

parent -- pointer to an object, it will be the parent of the new keyboard

Ritorna

pointer to the created keyboard

void **lv_keyboard_set_textarea**(lv_obj_t *kb, lv_obj_t *ta)

Assign a Text Area to the Keyboard. The pressed characters will be put there.

Parametri

- **kb** -- pointer to a Keyboard object
- **ta** -- pointer to a Text Area object to write there

void **lv_keyboard_set_mode**(lv_obj_t *kb, *lv_keyboard_mode_t* mode)

Set a new a mode (text or number map)

Parametri

- **kb** -- pointer to a Keyboard object
- **mode** -- the mode from '*lv_keyboard_mode_t*'

void **lv_keyboard_set_popovers**(lv_obj_t *kb, bool en)

Show the button title in a popover when pressed.

Parametri

- **kb** -- pointer to a Keyboard object
- **en** -- whether "popovers" mode is enabled

void **lv_keyboard_set_map**(lv_obj_t *kb, *lv_keyboard_mode_t* mode, const char *const map[], const *lv_buttonmatrix_ctrl_t* ctrl_map[])

Set a new map for the keyboard

Parametri

- **kb** -- pointer to a Keyboard object
- **mode** -- keyboard map to alter '*lv_keyboard_mode_t*'
- **map** -- pointer to a string array to describe the map. See '*lv_buttonmatrix_set_map()*' for more info.
- **ctrl_map** -- See '*lv_buttonmatrix_set_ctrl_map()*' for more info.

lv_obj_t ***lv_keyboard_get_textarea**(const lv_obj_t *kb)

Assign a Text Area to the Keyboard. The pressed characters will be put there.

Parametri

kb -- pointer to a Keyboard object

Ritorna

pointer to the assigned Text Area object

lv_keyboard_mode_t **lv_keyboard_get_mode**(const lv_obj_t *kb)

Set a new a mode (text or number map)

Parametri
kb -- pointer to a Keyboard object
Ritorna
the current mode from '*lv_keyboard_mode_t*'

bool **lv_keyboard_get_popovers**(const lv_obj_t *obj)

Tell whether "popovers" mode is enabled or not.
Parametri
obj -- pointer to a Keyboard object
Ritorna
true: "popovers" mode is enabled; false: disabled

const char *const ***lv_keyboard_get_map_array**(const lv_obj_t *kb)

Get the current map of a keyboard
Parametri
kb -- pointer to a keyboard object
Ritorna
the current map

uint32_t **lv_keyboard_get_selected_button**(const lv_obj_t *obj)

Get the index of the lastly "activated" button by the user (pressed, released, focused etc) Useful in the `event_cb` to get the text of the button, check if hidden etc.
Parametri
obj -- pointer to button matrix object
Ritorna
index of the last released button (LV_BUTTONMATRIX_BUTTON_NONE: if unset)

const char ***lv_keyboard_get_button_text**(const lv_obj_t *obj, uint32_t btn_id)

Get the button's text
Parametri
- **obj** -- pointer to button matrix object
- **btn_id** -- the index a button not counting new line characters.

Ritorna
text of btn_index` button

void **lv_keyboard_def_event_cb**(lv_event_t *e)

Default keyboard event to add characters to the Text area and change the map. If a custom `event_cb` is added to the keyboard this function can be called from it to handle the button clicks
Parametri
e -- the triggering event

Variables

const lv_obj_class_t **lv_keyboard_class**

lv_keyboard_private.h

struct **_lv_keyboard_t**

#include <lv_keyboard_private.h> Data of keyboard

Public Members

lv_buttonmatrix_t **btnm**

lv_obj_t ***ta**

Pointer to the assigned text area

lv_keyboard_mode_t **mode**

Key map type

uint8_t **popovers**

Show button titles in popovers on press

14.24.16 label

lv_label.h

Defines

LV_LABEL_DOT_NUM

LV_LABEL_POS_LAST

LV_LABEL_TEXT_SELECTION_OFF

LV_LABEL_DEFAULT_TEXT

Enums

enum **lv_label_long_mode_t**

Long mode behaviors. Used in 'lv_label_ext_t'

Values:

enumerator **LV_LABEL_LONG_MODE_WRAP**

Keep the object width, wrap lines longer than object width and expand the object height

enumerator **LV_LABEL_LONG_MODE_DOTS**

Keep the size and write dots at the end if the text is too long

enumerator **LV_LABEL_LONG_MODE_SCROLL**

Keep the size and roll the text back and forth

enumerator **LV_LABEL_LONG_MODE_SCROLL_CIRCULAR**

Keep the size and roll the text circularly

enumerator **LV_LABEL_LONG_MODE_CLIP**

Keep the size and clip the text out of it

enum **_lv_property_label_id_t**

Values:

enumerator **LV_PROPERTY_LABEL_TEXT**

enumerator **LV_PROPERTY_LABEL_LONG_MODE**

enumerator **LV_PROPERTY_LABEL_TEXT_SELECTION_START**

enumerator **LV_PROPERTY_LABEL_TEXT_SELECTION_END**

enumerator **LV_PROPERTY_LABEL_END**

Functions

lv_obj_t ***lv_label_create**(lv_obj_t *parent)

Create a label object

Parametri

parent -- pointer to an object, it will be the parent of the new label.

Ritorna

pointer to the created button

void **lv_label_set_text**(lv_obj_t *obj, const char *text)

Set a new text for a label. Memory will be allocated to store the text by the label.

Parametri

- **obj** -- pointer to a label object
- **text** -- '\0' terminated character string. NULL to refresh with the current text.

void **lv_label_set_text_fmt**(lv_obj_t *obj, const char *fmt, ...)

Set a new formatted text for a label. Memory will be allocated to store the text by the label.

Example:

```
lv_label_set_text_fmt(label1, "%d user", user_num);
```

Parametri

- **obj** -- pointer to a label object
- **fmt** -- `printf`-like format

void **lv_label_set_text_vfmt**(lv_obj_t *obj, const char *fmt, va_list args)

Set a new formatted text for a label. Memory will be allocated to store the text by the label.

Example:

```
va_list args;
va_start(args, fmt);
lv_label_set_text_vfmt(label1, fmt, args);
va_end(args);
```

Parametri

- **obj** -- pointer to a label object
- **fmt** -- `printf`-like format
- **args** -- variadic argments list

void **lv_label_set_text_static**(lv_obj_t *obj, const char *text)

Set a static text. It will not be saved by the label so the 'text' variable has to be 'alive' while the label exists.

Parametri

- **obj** -- pointer to a label object
- **text** -- pointer to a text. NULL to refresh with the current text.

void **lv_label_set_long_mode**(lv_obj_t *obj, *lv_label_long_mode_t* long_mode)

Set the behavior of the label with text longer than the object size

Parametri

- **obj** -- pointer to a label object

- **long_mode** -- the new mode from 'lv_label_long_mode' enum. In LV_LONG_WRAP/DOT/SCROLL/SCROLL_CIRC the size of the label should be set AFTER this function

void **lv_label_set_text_selection_start**(lv_obj_t *obj, uint32_t index)

Set where text selection should start

Parametri

- **obj** -- pointer to a label object
- **index** -- character index from where selection should start. `LV_LABEL_TEXT_SELECTION_OFF` for no selection

void **lv_label_set_text_selection_end**(lv_obj_t *obj, uint32_t index)

Set where text selection should end

Parametri

- **obj** -- pointer to a label object
- **index** -- character index where selection should end. `LV_LABEL_TEXT_SELECTION_OFF` for no selection

void **lv_label_set_recolor**(lv_obj_t *obj, bool en)

Enable the recoloring by in-line commands

Parametri

- **obj** -- pointer to a label object
- **en** -- true: enable recoloring, false: disable Example: "This is a #ff0000 red# word"

char ***lv_label_get_text**(const lv_obj_t *obj)

Get the text of a label

Parametri

obj -- pointer to a label object

Ritorna

the text of the label

lv_label_long_mode_t **lv_label_get_long_mode**(const lv_obj_t *obj)

Get the long mode of a label

Parametri

obj -- pointer to a label object

Ritorna

the current long mode

void **lv_label_get_letter_pos**(const lv_obj_t *obj, uint32_t char_id, *lv_point_t* *pos)

Get the relative x and y coordinates of a letter

Parametri

- **obj** -- pointer to a label object
- **char_id** -- index of the character [0 ... text length - 1]. Expressed in character index, not byte index (different in UTF-8)
- **pos** -- store the result here (E.g. index = 0 gives 0;0 coordinates if the text if aligned to the left)

uint32_t **lv_label_get_letter_on**(const lv_obj_t *obj, *lv_point_t* *pos_in, bool bidi)

Get the index of letter on a relative point of a label.

Parametri

- **obj** -- pointer to label object
- **pos_in** -- pointer to point with coordinates on a the label
- **bidi** -- whether to use bidi processed

Ritorna

The index of the letter on the 'pos_p' point (E.g. on 0;0 is the 0. letter if aligned to the left) Expressed in character index and not byte index (different in UTF-8)

bool **lv_label_is_char_under_pos**(const lv_obj_t *obj, *lv_point_t* *pos)

Check if a character is drawn under a point.

Parametri

- **obj** -- pointer to a label object
- **pos** -- Point to check for character under

Ritorna
whether a character is drawn under the point

uint32_t **lv_label_get_text_selection_start**(const lv_obj_t *obj)

Get the selection start index.

Parametri
obj -- pointer to a label object.

Ritorna
selection start index. `LV_LABEL_TEXT_SELECTION_OFF` if nothing is selected.

uint32_t **lv_label_get_text_selection_end**(const lv_obj_t *obj)

Get the selection end index.

Parametri
obj -- pointer to a label object.

Ritorna
selection end index. `LV_LABEL_TXT_SEL_OFF` if nothing is selected.

bool **lv_label_get_recolor**(const lv_obj_t *obj)

Get the recoloring attribute.

Parametri
obj -- pointer to a label object.

Ritorna
true: recoloring is enabled, false: recoloring is disabled

void **lv_label_ins_text**(lv_obj_t *obj, uint32_t pos, const char *txt)

Insert a text to a label. The label text cannot be static.

Parametri
- **obj** -- pointer to a label object
- **pos** -- character index to insert. Expressed in character index and not byte index. 0: before first char. LV_LABEL_POS_LAST: after last char.
- **txt** -- pointer to the text to insert

void **lv_label_cut_text**(lv_obj_t *obj, uint32_t pos, uint32_t cnt)

Delete characters from a label. The label text cannot be static.

Parametri
- **obj** -- pointer to a label object
- **pos** -- character index from where to cut. Expressed in character index and not byte index. 0: start in front of the first character
- **cnt** -- number of characters to cut

Variables

const lv_obj_class_t **lv_label_class**

lv_label_private.h

struct **_lv_label_t**

Public Members

lv_obj_t **obj**

char ***text**

char **dot**[3 + 1]

Bytes that have been replaced with dots

uint32_t **dot_begin**

Offset where bytes have been replaced with dots

lv_draw_label_hint_t **hint**

uint32_t **sel_start**

uint32_t **sel_end**

lv_point_t **size_cache**

Text size cache

lv_point_t **offset**

Text draw position offset

lv_label_long_mode_t **long_mode**

Determine what to do with the long texts

uint8_t **static_txt**

Flag to indicate the text is static

uint8_t **recolor**

Enable in-line letter re-coloring

uint8_t **expand**

Ignore real width (used by the library with LV_LABEL_LONG_MODE_SCROLL)

uint8_t **invalid_size_cache**

1: Recalculate size and update cache

lv_point_t **text_size**

14.24.17 led

lv_led.h

Defines

LV_LED_BRIGHT_MIN

Brightness when the LED if OFF

LV_LED_BRIGHT_MAX

Brightness when the LED if ON

Functions

lv_obj_t *`lv_led_create`(lv_obj_t *parent)

Create a led object

Parametri

parent -- pointer to an object, it will be the parent of the new led

Ritorna

pointer to the created led

void `lv_led_set_color`(lv_obj_t *led, *lv_color_t* color)

Set the color of the LED

Parametri

- **led** -- pointer to a LED object
- **color** -- the color of the LED

void `lv_led_set_brightness`(lv_obj_t *led, uint8_t bright)

Set the brightness of a LED object

Parametri

- **led** -- pointer to a LED object
- **bright** -- LV_LED_BRIGHT_MIN (max. dark) ... LV_LED_BRIGHT_MAX (max. light)

void `lv_led_on`(lv_obj_t *led)

Light on a LED

Parametri

led -- pointer to a LED object

void `lv_led_off`(lv_obj_t *led)

Light off a LED

Parametri

led -- pointer to a LED object

void `lv_led_toggle`(lv_obj_t *led)

Toggle the state of a LED

Parametri

led -- pointer to a LED object

uint8_t `lv_led_get_brightness`(const lv_obj_t *obj)

Get the brightness of a LED object

Parametri

obj -- pointer to LED object

Ritorna

bright 0 (max. dark) ... 255 (max. light)

Variables

const lv_obj_class_t `lv_led_class`

lv_led_private.h

struct `_lv_led_t`

#include <lv_led_private.h> Data of led

Public Members

lv_obj_t `obj`

lv_color_t **color**

uint8_t **bright**

Current brightness of the LED (0..255)

14.24.18 line

lv_line.h

Functions

lv_obj_t ***lv_line_create**(lv_obj_t *parent)

Create a line object

Parametri

parent -- pointer to an object, it will be the parent of the new line

Ritorna

pointer to the created line

void **lv_line_set_points**(lv_obj_t *obj, const *lv_point_precise_t* points[], uint32_t point_num)

Set an array of points. The line object will connect these points.

Parametri

- **obj** -- pointer to a line object
- **points** -- an array of points. Only the address is saved, so the array needs to be alive while the line exists
- **point_num** -- number of points in 'point_a'

void **lv_line_set_points_mutable**(lv_obj_t *obj, *lv_point_precise_t* points[], uint32_t point_num)

Set a non-const array of points. Identical to `lv_line_set_points` except the array may be retrieved by `lv_line_get_points_mutable`.

Parametri

- **obj** -- pointer to a line object
- **points** -- a non-const array of points. Only the address is saved, so the array needs to be alive while the line exists.
- **point_num** -- number of points in 'point_a'

void **lv_line_set_y_invert**(lv_obj_t *obj, bool en)

Enable (or disable) the y coordinate inversion. If enabled then y will be subtracted from the height of the object, therefore the y = 0 coordinate will be on the bottom.

Parametri

- **obj** -- pointer to a line object
- **en** -- true: enable the y inversion, false:disable the y inversion

const *lv_point_precise_t* ***lv_line_get_points**(lv_obj_t *obj)

Get the pointer to the array of points.

Parametri

obj -- pointer to a line object

Ritorna

const pointer to the array of points

uint32_t **lv_line_get_point_count**(lv_obj_t *obj)

Get the number of points in the array of points.

Parametri

obj -- pointer to a line object

Ritorna

number of points in array of points

bool **lv_line_is_point_array_mutable**(lv_obj_t *obj)

Check the mutability of the stored point array pointer.

Parametri
obj -- pointer to a line object
Ritorna
true: the point array pointer is mutable, false: constant

lv_point_precise_t ***lv_line_get_points_mutable**(lv_obj_t *obj)

Get a pointer to the mutable array of points or NULL if it is not mutable
Parametri
obj -- pointer to a line object
Ritorna
pointer to the array of points. NULL if not mutable.

bool **lv_line_get_y_invert**(const lv_obj_t *obj)

Get the y inversion attribute
Parametri
obj -- pointer to a line object
Ritorna
true: y inversion is enabled, false: disabled

Variables

const lv_obj_class_t **lv_line_class**

lv_line_private.h

struct **_lv_line_t**

#include <lv_line_private.h> Data of line

Public Members

lv_obj_t **obj**

const *lv_point_precise_t* ***constant**

lv_point_precise_t ***mut**

union *_lv_line_t* **point_array**

Pointer to an array with the points of the line

uint32_t **point_num**

Number of points in 'point_array'

uint32_t **y_inv**

1: y == 0 will be on the bottom

uint32_t **point_array_is_mutable**

whether the point array is const or mutable

14.24.19 list

lv_list.h

Functions

lv_obj_t *`lv_list_create`(lv_obj_t *parent)

Create a list object

Parametri

`parent` -- pointer to an object, it will be the parent of the new list

Ritorna

pointer to the created list

lv_obj_t *`lv_list_add_text`(lv_obj_t *list, const char *txt)

Add text to a list

Parametri

- `list` -- pointer to a list, it will be the parent of the new label
- `txt` -- text of the new label

Ritorna

pointer to the created label

lv_obj_t *`lv_list_add_button`(lv_obj_t *list, const void *icon, const char *txt)

Add button to a list

Parametri

- `list` -- pointer to a list, it will be the parent of the new button
- `icon` -- icon for the button, when NULL it will have no icon
- `txt` -- text of the new button, when NULL no text will be added

Ritorna

pointer to the created button

const char *`lv_list_get_button_text`(lv_obj_t *list, lv_obj_t *btn)

Get text of a given list button

Parametri

- `list` -- pointer to a list
- `btn` -- pointer to the button

Ritorna

text of btn, if btn doesn't have text "" will be returned

void `lv_list_set_button_text`(lv_obj_t *list, lv_obj_t *btn, const char *txt)

Set text of a given list button

Parametri

- `list` -- pointer to a list
- `btn` -- pointer to the button
- `txt` -- pointer to the text

Variables

const lv_obj_class_t `lv_list_class`

const lv_obj_class_t `lv_list_text_class`

const lv_obj_class_t `lv_list_button_class`

14.24.20 lottie

lv_lottie.h

Functions

lv_obj_t *`lv_lottie_create`(lv_obj_t *parent)

Create a lottie animation

Parametri

`parent` -- pointer to the parent widget

Ritorna
pointer to the created Lottie animation widget

void **lv_lottie_set_buffer**(lv_obj_t *obj, int32_t w, int32_t h, void *buf)

Set a buffer for the animation. It also defines the size of the animation

Parametri
- **obj** -- pointer to a lottie widget
- **w** -- width of the animation and buffer
- **h** -- height of the animation and buffer
- **buf** -- a static buffer with `width x height x 4` byte size

void **lv_lottie_set_draw_buf**(lv_obj_t *obj, lv_draw_buf_t *draw_buf)

Set a draw buffer for the animation. It also defines the size of the animation

Parametri
- **obj** -- pointer to a lottie widget
- **draw_buf** -- an initialized draw buffer with ARGB8888 color format

void **lv_lottie_set_src_data**(lv_obj_t *obj, const void *src, size_t src_size)

Set the source for the animation as an array

Parametri
- **obj** -- pointer to a lottie widget
- **src** -- the lottie animation converted to an nul terminated array
- **src_size** -- size of the source array in bytes

void **lv_lottie_set_src_file**(lv_obj_t *obj, const char *src)

Set the source for the animation as a path. Lottie doesn't use LVGL's File System API.

Parametri
- **obj** -- pointer to a lottie widget
- **src** -- path to a json file, e.g. "path/to/file.json"

lv_anim_t ***lv_lottie_get_anim**(lv_obj_t *obj)

Get the LVGL animation which controls the lottie animation

Parametri
obj -- pointer to a lottie widget

Ritorna
the LVGL animation

lv_lottie_private.h

struct **lv_lottie_t**

Public Members

lv_canvas_t **canvas**

Tvg_Paint ***tvg_paint**

Tvg_Canvas ***tvg_canvas**

Tvg_Animation ***tvg_anim**

lv_anim_t ***anim**

int32_t **last_rendered_time**

14.24.21 menu

lv_menu.h

Enums

enum **lv_menu_mode_header_t**

Values:

enumerator **LV_MENU_HEADER_TOP_FIXED**

Header is positioned at the top

enumerator **LV_MENU_HEADER_TOP_UNFIXED**

Header is positioned at the top and can be scrolled out of view

enumerator **LV_MENU_HEADER_BOTTOM_FIXED**

Header is positioned at the bottom

enum **lv_menu_mode_root_back_button_t**

Values:

enumerator **LV_MENU_ROOT_BACK_BUTTON_DISABLED**

enumerator **LV_MENU_ROOT_BACK_BUTTON_ENABLED**

Functions

lv_obj_t ***lv_menu_create**(lv_obj_t *parent)

Create a menu object

Parametri

parent -- pointer to an object, it will be the parent of the new menu

Ritorna

pointer to the created menu

lv_obj_t ***lv_menu_page_create**(lv_obj_t *menu, char const *const title)

Create a menu page object.

This call inserts the new page under menu->storage as its parent, which is itself a child of the menu, so the resulting object hierarchy is: menu => storage => new_page where `storage` is a Base Widget.

Parametri

- **menu** -- pointer to menu object.
- **title** -- pointer to text for title in header (NULL to not display title)

Ritorna

pointer to the created menu page

lv_obj_t ***lv_menu_cont_create**(lv_obj_t *parent)

Create a menu cont object

Parametri

parent -- pointer to a menu page object, it will be the parent of the new menu cont object

Ritorna

pointer to the created menu cont

lv_obj_t ***lv_menu_section_create**(lv_obj_t *parent)

Create a menu section object

Parametri

parent -- pointer to a menu page object, it will be the parent of the new menu section object

Ritorna
pointer to the created menu section

lv_obj_t *`lv_menu_separator_create`(lv_obj_t *parent)

Create a menu separator object

Parametri
`parent` -- pointer to a menu page object, it will be the parent of the new menu separator object

Ritorna
pointer to the created menu separator

void `lv_menu_set_page`(lv_obj_t *obj, lv_obj_t *page)

Set menu page to display in main

Parametri
- `obj` -- pointer to the menu
- `page` -- pointer to the menu page to set (NULL to clear main and clear menu history)

void `lv_menu_set_page_title`(lv_obj_t *page, char const *const title)

Set menu page title

Parametri
- `page` -- pointer to the menu page
- `title` -- pointer to text for title in header (NULL to not display title)

void `lv_menu_set_page_title_static`(lv_obj_t *page, char const *const title)

Set menu page title with a static text. It will not be saved by the label so the 'text' variable has to be 'alive' while the page exists.

Parametri
- `page` -- pointer to the menu page
- `title` -- pointer to text for title in header (NULL to not display title)

void `lv_menu_set_sidebar_page`(lv_obj_t *obj, lv_obj_t *page)

Set menu page to display in sidebar

Parametri
- `obj` -- pointer to the menu
- `page` -- pointer to the menu page to set (NULL to clear sidebar)

void `lv_menu_set_mode_header`(lv_obj_t *obj, *lv_menu_mode_header_t* mode)

Set the how the header should behave and its position

Parametri
- `obj` -- pointer to a menu
- `mode` -- LV_MENU_HEADER_TOP_FIXED/TOP_UNFIXED/BOTTOM_FIXED

void `lv_menu_set_mode_root_back_button`(lv_obj_t *obj, *lv_menu_mode_root_back_button_t* mode)

Set whether back button should appear at root

Parametri
- `obj` -- pointer to a menu
- `mode` -- LV_MENU_ROOT_BACK_BUTTON_DISABLED/ENABLED

void `lv_menu_set_load_page_event`(lv_obj_t *menu, lv_obj_t *obj, lv_obj_t *page)

Add menu to the menu item

Parametri
- `menu` -- pointer to the menu
- `obj` -- pointer to the obj
- `page` -- pointer to the page to load when obj is clicked

lv_obj_t *`lv_menu_get_cur_main_page`(lv_obj_t *obj)

Get a pointer to menu page that is currently displayed in main

Parametri
`obj` -- pointer to the menu

Ritorna
pointer to current page

lv_obj_t *`lv_menu_get_cur_sidebar_page`(lv_obj_t *obj)

Get a pointer to menu page that is currently displayed in sidebar

Parametri
`obj` -- pointer to the menu

Ritorna
pointer to current page

lv_obj_t *`lv_menu_get_main_header`(lv_obj_t *obj)

Get a pointer to main header obj

Parametri
`obj` -- pointer to the menu

Ritorna
pointer to main header obj

lv_obj_t *`lv_menu_get_main_header_back_button`(lv_obj_t *obj)

Get a pointer to main header back btn obj

Parametri
`obj` -- pointer to the menu

Ritorna
pointer to main header back btn obj

lv_obj_t *`lv_menu_get_sidebar_header`(lv_obj_t *obj)

Get a pointer to sidebar header obj

Parametri
`obj` -- pointer to the menu

Ritorna
pointer to sidebar header obj

lv_obj_t *`lv_menu_get_sidebar_header_back_button`(lv_obj_t *obj)

Get a pointer to sidebar header obj

Parametri
`obj` -- pointer to the menu

Ritorna
pointer to sidebar header back btn obj

bool `lv_menu_back_button_is_root`(lv_obj_t *menu, lv_obj_t *obj)

Check if an obj is a root back btn

Parametri
- `menu` -- pointer to the menu
- `obj` -- pointer to the back button

Ritorna
true if it is a root back btn

void `lv_menu_clear_history`(lv_obj_t *obj)

Clear menu history

Parametri
`obj` -- pointer to the menu

Variables

const lv_obj_class_t `lv_menu_class`

const lv_obj_class_t `lv_menu_page_class`

const lv_obj_class_t `lv_menu_cont_class`

const lv_obj_class_t **lv_menu_section_class**

const lv_obj_class_t **lv_menu_separator_class**

const lv_obj_class_t **lv_menu_sidebar_cont_class**

const lv_obj_class_t **lv_menu_main_cont_class**

const lv_obj_class_t **lv_menu_sidebar_header_cont_class**

const lv_obj_class_t **lv_menu_main_header_cont_class**

lv_menu_private.h

struct **_lv_menu_load_page_event_data_t**

Public Members

lv_obj_t ***menu**

lv_obj_t ***page**

struct **_lv_menu_history_t**

Public Members

lv_obj_t ***page**

struct **_lv_menu_t**

Public Members

lv_obj_t **obj**

lv_obj_t ***storage**

a pointer to obj that is the parent of all pages not displayed

lv_obj_t ***main**

lv_obj_t ***main_page**

lv_obj_t ***main_header**

lv_obj_t ***main_header_back_btn**

a pointer to obj that on click triggers back btn event handler, can be same as 'main_header'

lv_obj_t ***main_header_title**

lv_obj_t ***sidebar**

lv_obj_t ***sidebar_page**

lv_obj_t ***sidebar_header**

lv_obj_t ***sidebar_header_back_btn**

a pointer to obj that on click triggers back btn event handler, can be same as 'sidebar_header'

lv_obj_t ***sidebar_header_title**

lv_obj_t ***selected_tab**

lv_ll_t **history_ll**

uint8_t **cur_depth**

uint8_t **prev_depth**

uint8_t **sidebar_generated**

lv_menu_mode_header_t **mode_header**

lv_menu_mode_root_back_button_t **mode_root_back_btn**

struct **_lv_menu_page_t**

Public Members

lv_obj_t **obj**

char ***title**

bool **static_title**

14.24.22 msgbox

lv_msgbox.h

Functions

lv_obj_t ***lv_msgbox_create**(lv_obj_t *parent)

Create an empty message box

Parametri

parent -- the parent or NULL to create a modal msgbox

Ritorna

the created message box

lv_obj_t *`lv_msgbox_add_title`(lv_obj_t *obj, const char *title)

Add title to the message box. It also creates a header for the title.

Parametri

- `obj` -- pointer to a message box
- `title` -- the text of the tile

Ritorna

the created title label

lv_obj_t *`lv_msgbox_add_header_button`(lv_obj_t *obj, const void *icon)

Add a button to the header of to the message box. It also creates a header.

Parametri

- `obj` -- pointer to a message box
- `icon` -- the icon of the button

Ritorna

the created button

lv_obj_t *`lv_msgbox_add_text`(lv_obj_t *obj, const char *text)

Add a text to the content area of message box. Multiple texts will be created below each other.

Parametri

- `obj` -- pointer to a message box
- `text` -- text to add

Ritorna

the created button

lv_obj_t *`lv_msgbox_add_footer_button`(lv_obj_t *obj, const char *text)

Add a button to the footer of to the message box. It also creates a footer.

Parametri

- `obj` -- pointer to a message box
- `text` -- the text of the button

Ritorna

the created button

lv_obj_t *`lv_msgbox_add_close_button`(lv_obj_t *obj)

Add a close button to the message box. It also creates a header.

Parametri

`obj` -- pointer to a message box

Ritorna

the created close button

lv_obj_t *`lv_msgbox_get_header`(lv_obj_t *obj)

Get the header widget

Parametri

`obj` -- pointer to a message box

Ritorna

the header, or NULL if not exists

lv_obj_t *`lv_msgbox_get_footer`(lv_obj_t *obj)

Get the footer widget

Parametri

`obj` -- pointer to a message box

Ritorna

the footer, or NULL if not exists

lv_obj_t *`lv_msgbox_get_content`(lv_obj_t *obj)

Get the content widget

Parametri

`obj` -- pointer to a message box

Ritorna

the content

lv_obj_t *`lv_msgbox_get_title`(lv_obj_t *obj)

Get the title label

Parametri

obj -- pointer to a message box

Ritorna

the title, or NULL if it does not exist

void `lv_msgbox_close`(lv_obj_t *mbox)

Close a message box

Parametri

mbox -- pointer to a message box

void `lv_msgbox_close_async`(lv_obj_t *mbox)

Close a message box in the next call of the message box

Parametri

mbox -- pointer to a message box

Variables

const lv_obj_class_t `lv_msgbox_class`

const lv_obj_class_t `lv_msgbox_header_class`

const lv_obj_class_t `lv_msgbox_content_class`

const lv_obj_class_t `lv_msgbox_footer_class`

const lv_obj_class_t `lv_msgbox_header_button_class`

const lv_obj_class_t `lv_msgbox_footer_button_class`

const lv_obj_class_t `lv_msgbox_backdrop_class`

lv_msgbox_private.h

struct `_lv_msgbox_t`

Public Members

lv_obj_t `obj`

lv_obj_t *`header`

lv_obj_t *`content`

lv_obj_t *`footer`

lv_obj_t *`title`

14.24.23 objx_templ

lv_objx_templ.h

14.24.24 property

lv_obj_property_names.h

GENERATED FILE, DO NOT EDIT IT!

Variables

const lv_property_name_t **lv_animimage_property_names**[4]

const lv_property_name_t **lv_dropdown_property_names**[9]

const lv_property_name_t **lv_image_property_names**[11]

const lv_property_name_t **lv_keyboard_property_names**[4]

const lv_property_name_t **lv_label_property_names**[4]

const lv_property_name_t **lv_obj_property_names**[75]

const lv_property_name_t **lv_roller_property_names**[3]

const lv_property_name_t **lv_slider_property_names**[8]

const lv_property_name_t **lv_style_property_names**[120]

const lv_property_name_t **lv_textarea_property_names**[15]

lv_style_properties.h

GENERATED FILE, DO NOT EDIT IT!

Enums

enum **_lv_property_style_id_t**

Values:

enumerator **LV_PROPERTY_STYLE_ALIGN**

enumerator **LV_PROPERTY_STYLE_ANIM**

enumerator **LV_PROPERTY_STYLE_ANIM_DURATION**

enumerator **LV_PROPERTY_STYLE_ARC_COLOR**

enumerator **LV_PROPERTY_STYLE_ARC_IMAGE_SRC**

enumerator **LV_PROPERTY_STYLE_ARC_OPA**

enumerator **LV_PROPERTY_STYLE_ARC_ROUNDED**

enumerator **LV_PROPERTY_STYLE_ARC_WIDTH**

enumerator **LV_PROPERTY_STYLE_BASE_DIR**

enumerator **LV_PROPERTY_STYLE_BG_COLOR**

enumerator **LV_PROPERTY_STYLE_BG_GRAD**

enumerator **LV_PROPERTY_STYLE_BG_GRAD_COLOR**

enumerator **LV_PROPERTY_STYLE_BG_GRAD_DIR**

enumerator **LV_PROPERTY_STYLE_BG_GRAD_OPA**

enumerator **LV_PROPERTY_STYLE_BG_GRAD_STOP**

enumerator **LV_PROPERTY_STYLE_BG_IMAGE_OPA**

enumerator **LV_PROPERTY_STYLE_BG_IMAGE_RECOLOR**

enumerator **LV_PROPERTY_STYLE_BG_IMAGE_RECOLOR_OPA**

enumerator **LV_PROPERTY_STYLE_BG_IMAGE_SRC**

enumerator **LV_PROPERTY_STYLE_BG_IMAGE_TILED**

enumerator **LV_PROPERTY_STYLE_BG_MAIN_OPA**

enumerator **LV_PROPERTY_STYLE_BG_MAIN_STOP**

enumerator **LV_PROPERTY_STYLE_BG_OPA**

enumerator **LV_PROPERTY_STYLE_BITMAP_MASK_SRC**

enumerator **LV_PROPERTY_STYLE_BLEND_MODE**

enumerator **LV_PROPERTY_STYLE_BORDER_COLOR**

enumerator **LV_PROPERTY_STYLE_BORDER_OPA**

enumerator **LV_PROPERTY_STYLE_BORDER_POST**

enumerator **LV_PROPERTY_STYLE_BORDER_SIDE**

enumerator **LV_PROPERTY_STYLE_BORDER_WIDTH**

enumerator **LV_PROPERTY_STYLE_CLIP_CORNER**

enumerator **LV_PROPERTY_STYLE_COLOR_FILTER_DSC**

enumerator **LV_PROPERTY_STYLE_COLOR_FILTER_OPA**

enumerator **LV_PROPERTY_STYLE_FLEX_CROSS_PLACE**

enumerator **LV_PROPERTY_STYLE_FLEX_FLOW**

enumerator **LV_PROPERTY_STYLE_FLEX_GROW**

enumerator **LV_PROPERTY_STYLE_FLEX_MAIN_PLACE**

enumerator **LV_PROPERTY_STYLE_FLEX_TRACK_PLACE**

enumerator **LV_PROPERTY_STYLE_GRID_CELL_COLUMN_POS**

enumerator **LV_PROPERTY_STYLE_GRID_CELL_COLUMN_SPAN**

enumerator **LV_PROPERTY_STYLE_GRID_CELL_ROW_POS**

enumerator **LV_PROPERTY_STYLE_GRID_CELL_ROW_SPAN**

enumerator **LV_PROPERTY_STYLE_GRID_CELL_X_ALIGN**

enumerator **LV_PROPERTY_STYLE_GRID_CELL_Y_ALIGN**

enumerator **LV_PROPERTY_STYLE_GRID_COLUMN_ALIGN**

enumerator **LV_PROPERTY_STYLE_GRID_COLUMN_DSC_ARRAY**

enumerator **LV_PROPERTY_STYLE_GRID_ROW_ALIGN**

enumerator **LV_PROPERTY_STYLE_GRID_ROW_DSC_ARRAY**

enumerator **LV_PROPERTY_STYLE_HEIGHT**

enumerator **LV_PROPERTY_STYLE_IMAGE_OPA**

enumerator **LV_PROPERTY_STYLE_IMAGE_RECOLOR**

enumerator **LV_PROPERTY_STYLE_IMAGE_RECOLOR_OPA**

enumerator **LV_PROPERTY_STYLE_LAST_BUILT_IN_PROP**

enumerator **LV_PROPERTY_STYLE_LAYOUT**

enumerator **LV_PROPERTY_STYLE_LENGTH**

enumerator **LV_PROPERTY_STYLE_LINE_COLOR**

enumerator **LV_PROPERTY_STYLE_LINE_DASH_GAP**

enumerator **LV_PROPERTY_STYLE_LINE_DASH_WIDTH**

enumerator **LV_PROPERTY_STYLE_LINE_OPA**

enumerator **LV_PROPERTY_STYLE_LINE_ROUNDED**

enumerator **LV_PROPERTY_STYLE_LINE_WIDTH**

enumerator **LV_PROPERTY_STYLE_MARGIN_BOTTOM**

enumerator **LV_PROPERTY_STYLE_MARGIN_LEFT**

enumerator **LV_PROPERTY_STYLE_MARGIN_RIGHT**

enumerator **LV_PROPERTY_STYLE_MARGIN_TOP**

enumerator **LV_PROPERTY_STYLE_MAX_HEIGHT**

enumerator **LV_PROPERTY_STYLE_MAX_WIDTH**

enumerator **LV_PROPERTY_STYLE_MIN_HEIGHT**

enumerator **LV_PROPERTY_STYLE_MIN_WIDTH**

enumerator **LV_PROPERTY_STYLE_OPA**

enumerator **LV_PROPERTY_STYLE_OPA_LAYERED**

enumerator **LV_PROPERTY_STYLE_OUTLINE_COLOR**

enumerator **LV_PROPERTY_STYLE_OUTLINE_OPA**

enumerator **LV_PROPERTY_STYLE_OUTLINE_PAD**

enumerator **LV_PROPERTY_STYLE_OUTLINE_WIDTH**

enumerator **LV_PROPERTY_STYLE_PAD_BOTTOM**

enumerator **LV_PROPERTY_STYLE_PAD_COLUMN**

enumerator **LV_PROPERTY_STYLE_PAD_LEFT**

enumerator **LV_PROPERTY_STYLE_PAD_RADIAL**

enumerator **LV_PROPERTY_STYLE_PAD_RIGHT**

enumerator **LV_PROPERTY_STYLE_PAD_ROW**

enumerator **LV_PROPERTY_STYLE_PAD_TOP**

enumerator **LV_PROPERTY_STYLE_PROP_INV**

enumerator **LV_PROPERTY_STYLE_RADIAL_OFFSET**

enumerator **LV_PROPERTY_STYLE_RADIUS**

enumerator **LV_PROPERTY_STYLE_RECOLOR**

enumerator **LV_PROPERTY_STYLE_RECOLOR_OPA**

enumerator **LV_PROPERTY_STYLE_ROTARY_SENSITIVITY**

enumerator **LV_PROPERTY_STYLE_SHADOW_COLOR**

enumerator **LV_PROPERTY_STYLE_SHADOW_OFFSET_X**

enumerator **LV_PROPERTY_STYLE_SHADOW_OFFSET_Y**

enumerator **LV_PROPERTY_STYLE_SHADOW_OPA**

enumerator **LV_PROPERTY_STYLE_SHADOW_SPREAD**

enumerator **LV_PROPERTY_STYLE_SHADOW_WIDTH**

enumerator **LV_PROPERTY_STYLE_TEXT_ALIGN**

enumerator **LV_PROPERTY_STYLE_TEXT_COLOR**

enumerator **LV_PROPERTY_STYLE_TEXT_DECOR**

enumerator **LV_PROPERTY_STYLE_TEXT_FONT**

enumerator **LV_PROPERTY_STYLE_TEXT_LETTER_SPACE**

enumerator **LV_PROPERTY_STYLE_TEXT_LINE_SPACE**

enumerator **LV_PROPERTY_STYLE_TEXT_OPA**

enumerator **LV_PROPERTY_STYLE_TEXT_OUTLINE_STROKE_COLOR**

enumerator **LV_PROPERTY_STYLE_TEXT_OUTLINE_STROKE_OPA**

enumerator **LV_PROPERTY_STYLE_TEXT_OUTLINE_STROKE_WIDTH**

enumerator **LV_PROPERTY_STYLE_TRANSFORM_HEIGHT**

enumerator **LV_PROPERTY_STYLE_TRANSFORM_PIVOT_X**

enumerator **LV_PROPERTY_STYLE_TRANSFORM_PIVOT_Y**

enumerator **LV_PROPERTY_STYLE_TRANSFORM_ROTATION**

enumerator **LV_PROPERTY_STYLE_TRANSFORM_SCALE_X**

enumerator **LV_PROPERTY_STYLE_TRANSFORM_SCALE_Y**

enumerator **LV_PROPERTY_STYLE_TRANSFORM_SKEW_X**

enumerator **LV_PROPERTY_STYLE_TRANSFORM_SKEW_Y**

enumerator **LV_PROPERTY_STYLE_TRANSFORM_WIDTH**

enumerator **LV_PROPERTY_STYLE_TRANSITION**

enumerator **LV_PROPERTY_STYLE_TRANSLATE_RADIAL**

enumerator **LV_PROPERTY_STYLE_TRANSLATE_X**

enumerator **LV_PROPERTY_STYLE_TRANSLATE_Y**

enumerator **LV_PROPERTY_STYLE_WIDTH**

enumerator **LV_PROPERTY_STYLE_X**

enumerator **LV_PROPERTY_STYLE_Y**

14.24.25 roller

lv_roller.h

Enums

enum **lv_roller_mode_t**

Roller mode.

Values:

enumerator **LV_ROLLER_MODE_NORMAL**

Normal mode (roller ends at the end of the options).

enumerator **LV_ROLLER_MODE_INFINITE**

Infinite mode (roller can be scrolled forever).

enum **_lv_property_roller_id_t**

Values:

enumerator **LV_PROPERTY_ROLLER_OPTIONS**

enumerator **LV_PROPERTY_ROLLER_SELECTED**

enumerator **LV_PROPERTY_ROLLER_VISIBLE_ROW_COUNT**

enumerator **LV_PROPERTY_ROLLER_END**

Functions

lv_obj_t ***lv_roller_create**(lv_obj_t *parent)

Create a roller object

Parametri

parent -- pointer to an object, it will be the parent of the new roller.

Ritorna

pointer to the created roller

void **lv_roller_set_options**(lv_obj_t *obj, const char *options, *lv_roller_mode_t* mode)

Set the options on a roller

Parametri

- **obj** -- pointer to roller object
- **options** -- a string with '

  ' separated options. E.g. "One\nTwo\nThree"
- **mode** -- `LV_ROLLER_MODE_NORMAL` or `LV_ROLLER_MODE_INFINITE`

void **lv_roller_set_selected**(lv_obj_t *obj, uint32_t sel_opt, *lv_anim_enable_t* anim)

Set the selected option

Parametri

- **obj** -- pointer to a roller object
- **sel_opt** -- index of the selected option (0 ... number of option - 1);
- **anim** -- LV_ANIM_ON: set with animation; LV_ANIM_OFF set immediately

bool **lv_roller_set_selected_str**(lv_obj_t *obj, const char *sel_opt, *lv_anim_enable_t* anim)

Sets the given string as the selection on the roller. Does not alter the current selection on failure.

Parametri

- **obj** -- pointer to roller object
- **sel_opt** -- pointer to the string you want to set as an option
- **anim** -- LV_ANIM_ON: set with animation; LV_ANIM_OFF set immediately

Ritorna

`true` if set successfully and `false` if the given string does not exist as an option in the roller

void **lv_roller_set_visible_row_count**(lv_obj_t *obj, uint32_t row_cnt)

Set the height to show the given number of rows (options)

Parametri

- **obj** -- pointer to a roller object
- **row_cnt** -- number of desired visible rows

uint32_t **lv_roller_get_selected**(const lv_obj_t *obj)

Get the index of the selected option

Parametri

obj -- pointer to a roller object

Ritorna

index of the selected option (0 ... number of option - 1);

void **lv_roller_get_selected_str**(const lv_obj_t *obj, char *buf, uint32_t buf_size)

Get the current selected option as a string.

Parametri

- **obj** -- pointer to roller object
- **buf** -- pointer to an array to store the string
- **buf_size** -- size of `buf` in bytes. 0: to ignore it.

const char ***lv_roller_get_options**(const lv_obj_t *obj)

Get the options of a roller

Parametri

obj -- pointer to roller object

Ritorna

the options separated by '

'-s (E.g. "Option1\nOption2\nOption3")

uint32_t **lv_roller_get_option_count**(const lv_obj_t *obj)

Get the total number of options

Parametri

obj -- pointer to a roller object

Ritorna

the total number of options

Variables

const lv_obj_class_t **lv_roller_class**

lv_roller_private.h

struct **_lv_roller_t**

Public Members

lv_obj_t **obj**

uint32_t **option_cnt**

Number of options

uint32_t **sel_opt_id**

Index of the current option

uint32_t **sel_opt_id_ori**

Store the original index on focus

uint32_t **inf_page_cnt**

Number of extra pages added to make the roller look infinite

lv_roller_mode_t **mode**

uint32_t **moved**

14.24.26 scale

lv_scale.h

Defines

LV_SCALE_TOTAL_TICK_COUNT_DEFAULT

Default value of total minor ticks.

LV_SCALE_MAJOR_TICK_EVERY_DEFAULT

Default value of major tick every nth ticks.

LV_SCALE_LABEL_ENABLED_DEFAULT

Default value of scale label enabled.

LV_SCALE_LABEL_ROTATE_MATCH_TICKS

LV_SCALE_LABEL_ROTATE_KEEP_UPRIGHT

LV_SCALE_ROTATION_ANGLE_MASK

Enums

enum **lv_scale_mode_t**

Scale mode

Values:

enumerator **LV_SCALE_MODE_HORIZONTAL_TOP**

enumerator **LV_SCALE_MODE_HORIZONTAL_BOTTOM**

enumerator **LV_SCALE_MODE_VERTICAL_LEFT**

enumerator **LV_SCALE_MODE_VERTICAL_RIGHT**

enumerator **LV_SCALE_MODE_ROUND_INNER**

enumerator **LV_SCALE_MODE_ROUND_OUTER**

enumerator **LV_SCALE_MODE_LAST**

Functions

lv_obj_t ***lv_scale_create**(lv_obj_t *parent)

Create an scale object

Parametri

parent -- pointer to an object, it will be the parent of the new scale

Ritorna

pointer to created Scale Widget

void **lv_scale_set_mode**(lv_obj_t *obj, *lv_scale_mode_t* mode)

Set scale mode. See *lv_scale_mode_t*.

Parametri

- **obj** -- pointer to Scale Widget
- **mode** -- the new scale mode

void **lv_scale_set_total_tick_count**(lv_obj_t *obj, uint32_t total_tick_count)

Set scale total tick count (including minor and major ticks).

Parametri

- **obj** -- pointer to Scale Widget
- **total_tick_count** -- New total tick count

void **lv_scale_set_major_tick_every**(lv_obj_t *obj, uint32_t major_tick_every)

Sets how often major ticks are drawn.

Parametri

- **obj** -- pointer to Scale Widget
- **major_tick_every** -- the new count for major tick drawing

void **lv_scale_set_label_show**(lv_obj_t *obj, bool show_label)

Sets label visibility.

Parametri

- **obj** -- pointer to Scale Widget
- **show_label** -- true/false to enable tick label

void **lv_scale_set_range**(lv_obj_t *obj, int32_t min, int32_t max)

Set minimum and maximum values on Scale.

Parametri

- **obj** -- pointer to Scale Widget
- **min** -- minimum value of Scale
- **max** -- maximum value of Scale

void **lv_scale_set_min_value**(lv_obj_t *obj, int32_t min)

Set minimum values on Scale.

Parametri

- **obj** -- pointer to Scale Widget
- **min** -- minimum value of Scale

void **lv_scale_set_max_value**(lv_obj_t *obj, int32_t max)

Set maximum values on Scale.

Parametri

- **obj** -- pointer to Scale Widget
- **min** -- minimum value of Scale

void **lv_scale_set_angle_range**(lv_obj_t *obj, uint32_t angle_range)

Set angle between the low end and the high end of the Scale. (Applies only to round Scales.)

Parametri

- **obj** -- pointer to Scale Widget
- **max_angle** -- angle in degrees from Scale minimum where top end of Scale will be drawn

void **lv_scale_set_rotation**(lv_obj_t *obj, int32_t rotation)

Set angular offset from the 3-o'clock position of the low end of the Scale. (Applies only to round Scales.)

Parametri

- **obj** -- pointer to Scale Widget
- **rotation** -- clockwise angular offset (in degrees) from the 3-o'clock position of the low end of the scale; negative and >360 values are first normalized to range [0..360]. Examples:
  - 0 = 3 o'clock (right side)
  - 30 = 4 o'clock
  - 60 = 5 o'clock
  - 90 = 6 o'clock
  - 135 = midway between 7 and 8 o'clock (default)
  - 180 = 9 o'clock
  - 270 = 12 o'clock
  - 300 = 1 o'clock
  - 330 = 2 o'clock
  - -30 = 2 o'clock
  - 390 = 4 o'clock

void **lv_scale_set_line_needle_value**(lv_obj_t *obj, lv_obj_t *needle_line, int32_t needle_length, int32_t value)

Point line needle to specified value.

Parametri

- **obj** -- pointer to Scale Widget
- **needle_line** -- needle_line of the Scale. The line points will be allocated and managed by the Scale unless the line point array was previously set using `lv_line_set_points_mutable`.
- **needle_length** -- length of the needle
  - needle_length>0: needle_length=needle_length;
  - needle_length<0: needle_length=radius-|needle_length|;
- **value** -- Scale value needle will point to

void **lv_scale_set_image_needle_value**(lv_obj_t *obj, lv_obj_t *needle_img, int32_t value)

Point image needle to specified value; image must point to the right. E.g. -O------>

Parametri

- **obj** -- pointer to Scale Widget
- **needle_img** -- pointer to needle's Image
- **value** -- Scale value needle will point to

void **lv_scale_set_text_src**(lv_obj_t *obj, const char *txt_src[])

Set custom text source for major ticks labels.

Parametri

- **obj** -- pointer to Scale Widget

- **txt_src** -- pointer to an array of strings which will be display at major ticks; last element must be a NULL pointer.

void **lv_scale_set_post_draw**(lv_obj_t *obj, bool en)

Draw Scale after all its children are drawn.

Parametri

- **obj** -- pointer to Scale Widget
- **en** -- true: enable post draw

void **lv_scale_set_draw_ticks_on_top**(lv_obj_t *obj, bool en)

Draw Scale ticks on top of all other parts.

Parametri

- **obj** -- pointer to Scale Widget
- **en** -- true: enable draw ticks on top of all parts

lv_scale_section_t ***lv_scale_add_section**(lv_obj_t *obj)

Add a Section to specified Scale. Section will not be drawn until a valid range is set for it using *lv_scale_set_section_range()*.

Parametri

obj -- pointer to Scale Widget

Ritorna

pointer to new Section

void **lv_scale_section_set_range**(lv_scale_section_t *section, int32_t min, int32_t max)

DEPRECATED, use lv_scale_set_section_range instead. Set range for specified Scale Section

Parametri

- **section** -- pointer to Section
- **range_min** -- Section new minimum value
- **range_max** -- Section new maximum value

void **lv_scale_set_section_range**(lv_obj_t *scale, lv_scale_section_t *section, int32_t min, int32_t max)

Set the range of a scale section

Parametri

- **scale** -- pointer to scale
- **section** -- pointer to section
- **range_min** -- the section's new minimum value
- **range_max** -- the section's new maximum value

void **lv_scale_set_section_min_value**(lv_obj_t *scale, lv_scale_section_t *section, int32_t min)

Set the minimum value of a scale section

Parametri

- **scale** -- pointer to scale
- **section** -- pointer to section
- **min** -- the section's new minimum value

void **lv_scale_set_section_max_value**(lv_obj_t *scale, lv_scale_section_t *section, int32_t max)

Set the maximum value of a scale section

Parametri

- **scale** -- pointer to scale
- **section** -- pointer to section
- **max** -- the section's new maximum value

void **lv_scale_section_set_style**(lv_scale_section_t *section, *lv_part_t* part, *lv_style_t* *section_part_style)

DEPRECATED, use lv_scale_set_section_style_main/indicator/items instead. Set style for specified part of Section.

Parametri

- **section** -- pointer to Section
- **part** -- the part of the Scale the style will apply to, e.g. LV_PART_INDICATOR

- **section_part_style** -- pointer to style to apply

void **lv_scale_set_section_style_main**(lv_obj_t *scale, lv_scale_section_t *section, const *lv_style_t* *style)

Set the style of the line on a section.

Parametri

- **scale** -- pointer to scale
- **section** -- pointer to section
- **style** -- point to a style

void **lv_scale_set_section_style_indicator**(lv_obj_t *scale, lv_scale_section_t *section, const *lv_style_t* *style)

Set the style of the major ticks and label on a section.

Parametri

- **scale** -- pointer to scale
- **section** -- pointer to section
- **style** -- point to a style

void **lv_scale_set_section_style_items**(lv_obj_t *scale, lv_scale_section_t *section, const *lv_style_t* *style)

Set the style of the minor ticks on a section.

Parametri

- **scale** -- pointer to scale
- **section** -- pointer to section
- **style** -- point to a style

lv_scale_mode_t **lv_scale_get_mode**(lv_obj_t *obj)

Get scale mode. See *lv_scale_mode_t*

Parametri

obj -- pointer to Scale Widget

Ritorna

Scale mode

int32_t **lv_scale_get_total_tick_count**(lv_obj_t *obj)

Get scale total tick count (including minor and major ticks)

Parametri

obj -- pointer to Scale Widget

Ritorna

Scale total tick count

int32_t **lv_scale_get_major_tick_every**(lv_obj_t *obj)

Get how often the major tick will be drawn

Parametri

obj -- pointer to Scale Widget

Ritorna

Scale major tick every count

int32_t **lv_scale_get_rotation**(lv_obj_t *obj)

Get angular location of low end of Scale.

Parametri

obj -- pointer to Scale Widget

Ritorna

Scale low end angular location

bool **lv_scale_get_label_show**(lv_obj_t *obj)

Gets label visibility

Parametri

obj -- pointer to Scale Widget

Ritorna

true if tick label is enabled, false otherwise

uint32_t **lv_scale_get_angle_range**(lv_obj_t *obj)

Get Scale's range in degrees

Parametri

obj -- pointer to Scale Widget

Ritorna

Scale's angle_range

int32_t **lv_scale_get_range_min_value**(lv_obj_t *obj)

Get minimum value for Scale

Parametri

obj -- pointer to Scale Widget

Ritorna

Scale's minimum value

int32_t **lv_scale_get_range_max_value**(lv_obj_t *obj)

Get maximum value for Scale

Parametri

obj -- pointer to Scale Widget

Ritorna

Scale's maximum value

Variables

const lv_obj_class_t **lv_scale_class**

lv_scale_private.h

struct **_lv_scale_section_t**

Public Members

const *lv_style_t* ***main_style**

Style to use for MAIN part(s) of scale when it falls within this section's range

const *lv_style_t* ***indicator_style**

Style to use for INDICATOR part(s) of scale when it falls within this section's range

const *lv_style_t* ***items_style**

Style to use for ITEMS part(s) of scale when it falls within this section's range

int32_t **range_min**

Scale parts with value >= this value will be drawn using applicable style.

int32_t **range_max**

Scale parts with value <= this value will be drawn using applicable style.

uint32_t **first_tick_idx_in_section**

Internal (set during drawing): Tick index of first tick that falls within this section; LV_SCALE_TICK_IDX_DEFAULT_ID if section contains no ticks.

uint32_t **last_tick_idx_in_section**

Internal (set during drawing): Tick index of last tick that falls within this section; LV_SCALE_TICK_IDX_DEFAULT_ID if section contains no ticks.

int32_t **first_tick_in_section_width**

Internal (set during drawing)

int32_t **last_tick_in_section_width**

Internal (set during drawing)

lv_point_t **first_tick_in_section**

Internal (set during drawing)

lv_point_t **last_tick_in_section**

Internal (set during drawing)

uint32_t **first_tick_idx_is_major**

Internal (set during drawing): true if `first_tick_idx_in_section` represents a major tick.

uint32_t **last_tick_idx_is_major**

Internal (set during drawing): true if `last_tick_idx_in_section` represents a major tick.

struct **_lv_scale_t**

Public Members

lv_obj_t **obj**

Base Widget part of Scale

lv_ll_t **section_ll**

Linked list for the sections (stores lv_scale_section_t)

const char ****txt_src**

Optional list of text strings for major ticks when custom labels are provided.

lv_scale_mode_t **mode**

Orientation and layout of scale.

int32_t **range_min**

Scale's minimum value

int32_t **range_max**

Scale's maximum value

uint32_t **total_tick_count**

Total number of ticks (major and minor)

uint32_t **major_tick_every**

Frequency of major ticks to minor ticks

uint32_t **label_enabled**

Draw labels for major ticks?

uint32_t **post_draw**

false: drawing occurs during LV_EVENT_DRAW_MAIN; true : drawing occurs during LV_EVENT_DRAW_POST.

uint32_t **draw_ticks_on_top**

Draw ticks on top of main line?

uint32_t **angle_range**

Degrees between low end and high end of scale

int32_t **rotation**

Clockwise angular offset from 3-o'clock position of low end of scale

int32_t **custom_label_cnt**

Number of custom labels provided in `txt_src`

int32_t **last_tick_width**

Width of last tick in pixels

int32_t **first_tick_width**

Width of first tick in pixels

14.24.27 slider

lv_slider.h

Enums

enum **lv_slider_mode_t**

Values:

enumerator **LV_SLIDER_MODE_NORMAL**

enumerator **LV_SLIDER_MODE_SYMMETRICAL**

enumerator **LV_SLIDER_MODE_RANGE**

enum **lv_slider_orientation_t**

Values:

enumerator **LV_SLIDER_ORIENTATION_AUTO**

enumerator **LV_SLIDER_ORIENTATION_HORIZONTAL**

enumerator **LV_SLIDER_ORIENTATION_VERTICAL**

enum **_lv_property_slider_id_t**

Values:

enumerator **LV_PROPERTY_SLIDER_VALUE**

enumerator **LV_PROPERTY_SLIDER_LEFT_VALUE**

enumerator **LV_PROPERTY_SLIDER_RANGE**

enumerator **LV_PROPERTY_SLIDER_MIN_VALUE**

enumerator **LV_PROPERTY_SLIDER_MAX_VALUE**

enumerator **LV_PROPERTY_SLIDER_MODE**

enumerator **LV_PROPERTY_SLIDER_IS_DRAGGED**

enumerator **LV_PROPERTY_SLIDER_IS_SYMMETRICAL**

enumerator **LV_PROPERTY_SLIDER_END**

Functions

lv_obj_t ***lv_slider_create**(lv_obj_t *parent)

Create a slider object

Parametri

parent -- pointer to an object, it will be the parent of the new slider.

Ritorna

pointer to the created slider

void **lv_slider_set_value**(lv_obj_t *obj, int32_t value, *lv_anim_enable_t* anim)

Set a new value on the slider

Parametri

- **obj** -- pointer to a slider object
- **value** -- the new value
- **anim** -- LV_ANIM_ON: set the value with an animation; LV_ANIM_OFF: change the value immediately

void **lv_slider_set_start_value**(lv_obj_t *obj, int32_t value, *lv_anim_enable_t* anim)

Set a new value for the left knob of a slider

Parametri

- **obj** -- pointer to a slider object
- **value** -- new value
- **anim** -- LV_ANIM_ON: set the value with an animation; LV_ANIM_OFF: change the value immediately

void **lv_slider_set_range**(lv_obj_t *obj, int32_t min, int32_t max)

Set the minimum and the maximum values of a bar

Parametri

- **obj** -- pointer to the slider object
- **min** -- minimum value
- **max** -- maximum value

void **lv_slider_set_min_value**(lv_obj_t *obj, int32_t min)

Set the minimum values of a bar

Parametri

- **obj** -- pointer to the slider object

- **min** -- minimum value

void **lv_slider_set_max_value**(lv_obj_t *obj, int32_t max)

Set the maximum values of a bar

Parametri

- **obj** -- pointer to the slider object
- **max** -- maximum value

void **lv_slider_set_mode**(lv_obj_t *obj, *lv_slider_mode_t* mode)

Set the mode of slider.

Parametri

- **obj** -- pointer to a slider object
- **mode** -- the mode of the slider. See `lv_slider_mode_t`

void **lv_slider_set_orientation**(lv_obj_t *obj, *lv_slider_orientation_t* orientation)

Set the orientation of slider.

Parametri

- **obj** -- pointer to a slider object
- **orientation** -- slider orientation from `lv_slider_orientation_t`

int32_t **lv_slider_get_value**(const lv_obj_t *obj)

Get the value of the main knob of a slider

Parametri

obj -- pointer to a slider object

Ritorna

the value of the main knob of the slider

int32_t **lv_slider_get_left_value**(const lv_obj_t *obj)

Get the value of the left knob of a slider

Parametri

obj -- pointer to a slider object

Ritorna

the value of the left knob of the slider

int32_t **lv_slider_get_min_value**(const lv_obj_t *obj)

Get the minimum value of a slider

Parametri

obj -- pointer to a slider object

Ritorna

the minimum value of the slider

int32_t **lv_slider_get_max_value**(const lv_obj_t *obj)

Get the maximum value of a slider

Parametri

obj -- pointer to a slider object

Ritorna

the maximum value of the slider

bool **lv_slider_is_dragged**(const lv_obj_t *obj)

Give the slider is being dragged or not

Parametri

obj -- pointer to a slider object

Ritorna

true: drag in progress false: not dragged

lv_slider_mode_t **lv_slider_get_mode**(lv_obj_t *slider)

Get the mode of the slider.

Parametri

slider -- pointer to a slider object

Ritorna

see `lv_slider_mode_t`

lv_slider_orientation_t **lv_slider_get_orientation**(lv_obj_t *slider)

Get the orientation of slider.

Parametri

obj -- pointer to a slider object

Ritorna

slider orientation from `lv_slider_orientation_t`

bool **lv_slider_is_symmetrical**(lv_obj_t *obj)

Give the slider is in symmetrical mode or not

Parametri

obj -- pointer to slider object

Ritorna

true: in symmetrical mode false : not in

Variables

const lv_obj_class_t **lv_slider_class**

lv_slider_private.h

struct **_lv_slider_t**

Public Members

lv_bar_t **bar**

Add the ancestor's type first

lv_area_t **left_knob_area**

lv_area_t **right_knob_area**

lv_point_t **pressed_point**

int32_t ***value_to_set**

Which bar value to set

uint8_t **dragging**

1: the slider is being dragged

uint8_t **left_knob_focus**

1: with encoder now the right knob can be adjusted

14.24.28 span

lv_span.h

Typedef

typedef struct *_lv_span_coords_t* **lv_span_coords_t**

Coords of a span

Enums

enum **lv_span_overflow_t**

Values:

enumerator **LV_SPAN_OVERFLOW_CLIP**

enumerator **LV_SPAN_OVERFLOW_ELLIPSIS**

enumerator **LV_SPAN_OVERFLOW_LAST**

Fence member

enum **lv_span_mode_t**

Values:

enumerator **LV_SPAN_MODE_FIXED**

fixed the obj size

enumerator **LV_SPAN_MODE_EXPAND**

Expand the object size to the text size

enumerator **LV_SPAN_MODE_BREAK**

Keep width, break the too long lines and expand height

enumerator **LV_SPAN_MODE_LAST**

Fence member

Functions

void **lv_span_stack_init**(void)

void **lv_span_stack_deinit**(void)

lv_obj_t ***lv_spangroup_create**(lv_obj_t *parent)

Create a spangroup object

Parametri

parent -- pointer to an object, it will be the parent of the new spangroup

Ritorna

pointer to the created spangroup

lv_span_t ***lv_spangroup_add_span**(lv_obj_t *obj)

Create a span string descriptor and add to spangroup.

Parametri

obj -- pointer to a spangroup object.

Ritorna

pointer to the created span.

void **lv_spangroup_delete_span**(lv_obj_t *obj, lv_span_t *span)

Remove the span from the spangroup and free memory.

Parametri

- **obj** -- pointer to a spangroup object.
- **span** -- pointer to a span.

void **lv_span_set_text**(lv_span_t *span, const char *text)

Set a new text for a span. Memory will be allocated to store the text by the span. As the spangroup is not passed a redraw (invalidation) can't be triggered automatically. Therefore `lv_spangroup_refresh(spangroup)` needs to be called manually,

Parametri

- **span** -- pointer to a span.
- **text** -- pointer to a text.

void **lv_span_set_text_static**(lv_span_t *span, const char *text)

Set a static text. It will not be saved by the span so the 'text' variable has to be 'alive' while the span exist. As the spangroup is not passed a redraw (invalidation) can't be triggered automatically. Therefore `lv_spangroup_refresh(spangroup)` needs to be called manually,

Set a static text. It will not be saved by the span so the 'text' variable has to be 'alive' while the span exist.

Parametri

- **span** -- pointer to a span.
- **text** -- pointer to a text.
- **span** -- pointer to a span.
- **text** -- pointer to a text.

void **lv_spangroup_set_span_text**(lv_obj_t *obj, lv_span_t *span, const char *text)

Set a new text for a span. Memory will be allocated to store the text by the span.

Parametri

- **obj** -- pointer to a spangroup widget.
- **span** -- pointer to a span.
- **text** -- pointer to a text.

void **lv_spangroup_set_span_text_static**(lv_obj_t *obj, lv_span_t *span, const char *text)

Set a new text for a span. Memory will be allocated to store the text by the span.

Parametri

- **obj** -- pointer to a spangroup widget.
- **span** -- pointer to a span.
- **text** -- pointer to a text.

void **lv_spangroup_set_span_style**(lv_obj_t *obj, lv_span_t *span, const *lv_style_t* *style)

Copy all style properties of style to the bbuilt-in static style of the span.

Parametri

- **obj** -- pointer_to a spangroup
- **span** -- pointer to a span.
- **style** -- pointer to a style to copy into the span's built-in style

void **lv_spangroup_set_align**(lv_obj_t *obj, *lv_text_align_t* align)

DEPRECATED. Use the text_align style property instead Set the align of the spangroup.

Parametri

- **obj** -- pointer to a spangroup object.
- **align** -- see *lv_text_align_t* for details.

void **lv_spangroup_set_overflow**(lv_obj_t *obj, *lv_span_overflow_t* overflow)

Set the overflow of the spangroup.

Parametri

- **obj** -- pointer to a spangroup object.
- **overflow** -- see *lv_span_overflow_t* for details.

void **lv_spangroup_set_indent**(lv_obj_t *obj, int32_t indent)

Set the indent of the spangroup.

Parametri

- **obj** -- pointer to a spangroup object.
- **indent** -- the first line indentation

void **lv_spangroup_set_mode**(lv_obj_t *obj, *lv_span_mode_t* mode)

DEPRECATED, set the width to LV_SIZE_CONTENT or fixed value to control expanding/wrapping" Set the mode of the spangroup.

Parametri

- **obj** -- pointer to a spangroup object.
- **mode** -- see *lv_span_mode_t* for details.

void **lv_spangroup_set_max_lines**(lv_obj_t *obj, int32_t lines)

Set maximum lines of the spangroup.

Parametri

- **obj** -- pointer to a spangroup object.
- **lines** -- max lines that can be displayed in LV_SPAN_MODE_BREAK mode. < 0 means no limit.

lv_style_t ***lv_span_get_style**(lv_span_t *span)

Get a pointer to the style of a span's built-in style. Any lv_style_set_... functions can be applied on the returned style.

Parametri

span -- pointer to the span

Ritorna

pointer to the style. (valid as long as the span is valid)

const char ***lv_span_get_text**(lv_span_t *span)

Get a pointer to the text of a span

Parametri

span -- pointer to the span

Ritorna

pointer to the text

lv_span_t ***lv_spangroup_get_child**(const lv_obj_t *obj, int32_t id)

Get a spangroup child by its index.

Parametri

- **obj** -- The spangroup object
- **id** -- the index of the child. 0: the oldest (firstly created) child 1: the second oldest child count-1: the youngest -1: the youngest -2: the second youngest

Ritorna

The child span at index `id`, or NULL if the ID does not exist

uint32_t **lv_spangroup_get_span_count**(const lv_obj_t *obj)

Get number of spans

Parametri

obj -- the spangroup object to get the child count of.

Ritorna

the span count of the spangroup.

lv_text_align_t **lv_spangroup_get_align**(lv_obj_t *obj)

Get the align of the spangroup.

Parametri

obj -- pointer to a spangroup object.

Ritorna

the align value.

lv_span_overflow_t **lv_spangroup_get_overflow**(lv_obj_t *obj)

Get the overflow of the spangroup.

Parametri

obj -- pointer to a spangroup object.

Ritorna

the overflow value.

int32_t **lv_spangroup_get_indent**(lv_obj_t *obj)

Get the indent of the spangroup.

Parametri

obj -- pointer to a spangroup object.

Ritorna

the indent value.

lv_span_mode_t **lv_spangroup_get_mode**(lv_obj_t *obj)

Get the mode of the spangroup.

Parametri

obj -- pointer to a spangroup object.

int32_t **lv_spangroup_get_max_lines**(lv_obj_t *obj)

Get maximum lines of the spangroup.

Parametri

obj -- pointer to a spangroup object.

Ritorna

the max lines value.

int32_t **lv_spangroup_get_max_line_height**(lv_obj_t *obj)

Get max line height of all span in the spangroup.

Parametri

obj -- pointer to a spangroup object.

uint32_t **lv_spangroup_get_expand_width**(lv_obj_t *obj, uint32_t max_width)

Get the text content width when all span of spangroup on a line.

Parametri

- **obj** -- pointer to a spangroup object.
- **max_width** -- if text content width >= max_width, return max_width to reduce computation, if max_width == 0, returns the text content width.

Ritorna

text content width or max_width.

int32_t **lv_spangroup_get_expand_height**(lv_obj_t *obj, int32_t width)

Get the text content height with width fixed.

Parametri

- **obj** -- pointer to a spangroup object.
- **width** -- the width of the span group.

lv_span_coords_t **lv_spangroup_get_span_coords**(lv_obj_t *obj, const lv_span_t *span)

Get the span's coords in the spangroup.

Nota

Before calling this function, please make sure that the layout of span group has been updated. Like calling *lv_obj_update_layout()* like function.

```
+--------+
|Heading +--->------------------+
|  Pos   |   |     Heading      |
+--------+---+------------------+
|                               |
|                               |
|                               |
|            Middle   +--------+|
|                     |Trailing||
|                  +-|  Pos   ||
|                  | +--------+|
+------------------v-----------+
|     Trailing      |
+-------------------+
```

Parametri
- **obj** -- pointer to a spangroup object.
- **span** -- pointer to a span.

Ritorna
the span's coords in the spangroup.

lv_span_t ***lv_spangroup_get_span_by_point**(lv_obj_t *obj, const *lv_point_t* *point)

Get the span object by point.

Parametri
- **obj** -- pointer to a spangroup object.
- **point** -- pointer to point containing absolute coordinates

Ritorna
pointer to the span under the point or `NULL` if not found.

void **lv_spangroup_refresh**(lv_obj_t *obj)

Update the mode of the spangroup.

Parametri
obj -- pointer to a spangroup object.

Variables

const lv_obj_class_t **lv_spangroup_class**

struct **_lv_span_coords_t**

#include <lv_span.h> Coords of a span

Public Members

lv_area_t **heading**

lv_area_t **middle**

lv_area_t **trailing**

lv_span_private.h

struct **_lv_span_t**

Public Members

char ***txt**

a pointer to display text

lv_style_t **style**

display text style

uint32_t **static_flag**

the text is static flag

lv_point_t **trailing_pos**

int32_t **trailing_height**

struct **_lv_spangroup_t**

#include <lv_span_private.h> Data of label

Public Members

lv_obj_t **obj**

int32_t **lines**

int32_t **indent**

first line indent

int32_t **cache_w**

the cache automatically calculates the width

int32_t **cache_h**

similar cache_w

lv_ll_t **child_ll**

uint32_t **overflow**

details see *lv_span_overflow_t*

uint32_t **refresh**

the spangroup need refresh cache_w and cache_h

14.24.29 spinbox

lv_spinbox.h

Defines

LV_SPINBOX_MAX_DIGIT_COUNT

Functions

lv_obj_t ***lv_spinbox_create**(lv_obj_t *parent)

Create a spinbox object

Parametri

parent -- pointer to an object, it will be the parent of the new spinbox

Ritorna

pointer to the created spinbox

void **lv_spinbox_set_value**(lv_obj_t *obj, int32_t v)

Set spinbox value

Parametri

- **obj** -- pointer to spinbox
- **v** -- value to be set

void **lv_spinbox_set_rollover**(lv_obj_t *obj, bool rollover)

Set spinbox rollover function

Parametri

- **obj** -- pointer to spinbox
- **rollover** -- true or false to enable or disable (default)

void **lv_spinbox_set_digit_format**(lv_obj_t *obj, uint32_t digit_count, uint32_t sep_pos)

Set spinbox digit format (digit count and decimal format)

Parametri

- **obj** -- pointer to spinbox
- **digit_count** -- number of digit excluding the decimal separator and the sign
- **sep_pos** -- number of digit before the decimal point. If 0, decimal point is not shown

void **lv_spinbox_set_step**(lv_obj_t *obj, uint32_t step)

Set spinbox step

Parametri

- **obj** -- pointer to spinbox
- **step** -- steps on increment/decrement. Can be 1, 10, 100, 1000, etc the digit that will change.

void **lv_spinbox_set_range**(lv_obj_t *obj, int32_t range_min, int32_t range_max)

Set spinbox value range

Parametri

- **obj** -- pointer to spinbox
- **range_min** -- maximum value, inclusive
- **range_max** -- minimum value, inclusive

void **lv_spinbox_set_cursor_pos**(lv_obj_t *obj, uint32_t pos)

Set cursor position to a specific digit for edition

Parametri

- **obj** -- pointer to spinbox
- **pos** -- selected position in spinbox

void **lv_spinbox_set_digit_step_direction**(lv_obj_t *obj, *lv_dir_t* direction)

Set direction of digit step when clicking an encoder button while in editing mode

Parametri

- **obj** -- pointer to spinbox
- **direction** -- the direction (LV_DIR_RIGHT or LV_DIR_LEFT)

bool **lv_spinbox_get_rollover**(lv_obj_t *obj)

Get spinbox rollover function status

Parametri

obj -- pointer to spinbox

int32_t **lv_spinbox_get_value**(lv_obj_t *obj)

Get the spinbox numeral value (user has to convert to float according to its digit format)

Parametri

obj -- pointer to spinbox

Ritorna

value integer value of the spinbox

int32_t **lv_spinbox_get_step**(lv_obj_t *obj)

Get the spinbox step value (user has to convert to float according to its digit format)

Parametri

obj -- pointer to spinbox

Ritorna

value integer step value of the spinbox

void **lv_spinbox_step_next**(lv_obj_t *obj)

Select next lower digit for edition by dividing the step by 10

Parametri

obj -- pointer to spinbox

void **lv_spinbox_step_prev**(lv_obj_t *obj)

Select next higher digit for edition by multiplying the step by 10

Parametri

obj -- pointer to spinbox

void **lv_spinbox_increment**(lv_obj_t *obj)

Increment spinbox value by one step

Parametri

obj -- pointer to spinbox

void **lv_spinbox_decrement**(lv_obj_t *obj)

Decrement spinbox value by one step

Parametri

obj -- pointer to spinbox

Variables

const lv_obj_class_t **lv_spinbox_class**

lv_spinbox_private.h

struct **_lv_spinbox_t**

#include <lv_spinbox_private.h> Data of spinbox

Public Members

lv_textarea_t **ta**

Ext. of ancestor

int32_t **value**

int32_t **range_max**

int32_t **range_min**

int32_t **step**

uint32_t **digit_count**

uint32_t **dec_point_pos**

if 0, there is no separator and the number is an integer

uint32_t **rollover**

Set to true for rollover functionality

uint32_t **digit_step_dir**

the direction the digit will step on encoder button press when editing

14.24.30 spinner

lv_spinner.h

Functions

lv_obj_t ***lv_spinner_create**(lv_obj_t *parent)

Create a spinner widget

Parametri

parent -- pointer to an object, it will be the parent of the new spinner.

Ritorna

the created spinner

void **lv_spinner_set_anim_params**(lv_obj_t *obj, uint32_t t, uint32_t angle)

Set the animation time and arc length of the spinner

Parametri

- **obj** -- pointer to a spinner
- **t** -- the animation time in milliseconds
- **angle** -- the angle of the arc in degrees

Variables

const lv_obj_class_t **lv_spinner_class**

14.24.31 switch

lv_switch.h

Defines

LV_SWITCH_KNOB_EXT_AREA_CORRECTION

Switch knob extra area correction factor

Enums

enum **lv_switch_orientation_t**

Values:

enumerator **LV_SWITCH_ORIENTATION_AUTO**

enumerator **LV_SWITCH_ORIENTATION_HORIZONTAL**

enumerator **LV_SWITCH_ORIENTATION_VERTICAL**

Functions

lv_obj_t ***lv_switch_create**(lv_obj_t *parent)

Create a switch object

Parametri

parent -- pointer to an object, it will be the parent of the new switch

Ritorna

pointer to the created switch

void **lv_switch_set_orientation**(lv_obj_t *obj, *lv_switch_orientation_t* orientation)

Set the orientation of switch.

Parametri

- **obj** -- pointer to switch object
- **orientation** -- switch orientation from `lv_switch_orientation_t`

lv_switch_orientation_t **lv_switch_get_orientation**(lv_obj_t *obj)

Get the orientation of switch.

Parametri

obj -- pointer to switch object

Ritorna

switch orientation from ::lv_switch_orientation_t

Variables

const lv_obj_class_t **lv_switch_class**

lv_switch_private.h

struct **_lv_switch_t**

Public Members

lv_obj_t **obj**

int32_t **anim_state**

lv_switch_orientation_t **orientation**

Orientation of switch

14.24.32 table

lv_table.h

Defines

LV_TABLE_CELL_NONE

Enums

enum **lv_table_cell_ctrl_t**

Values:

enumerator **LV_TABLE_CELL_CTRL_NONE**

enumerator **LV_TABLE_CELL_CTRL_MERGE_RIGHT**

enumerator **LV_TABLE_CELL_CTRL_TEXT_CROP**

enumerator **LV_TABLE_CELL_CTRL_CUSTOM_1**

enumerator **LV_TABLE_CELL_CTRL_CUSTOM_2**

enumerator **LV_TABLE_CELL_CTRL_CUSTOM_3**

enumerator **LV_TABLE_CELL_CTRL_CUSTOM_4**

Functions

lv_obj_t ***lv_table_create**(lv_obj_t *parent)

Create a table object

Parametri

parent -- pointer to an object, it will be the parent of the new table

Ritorna

pointer to the created table

void **lv_table_set_cell_value**(lv_obj_t *obj, uint32_t row, uint32_t col, const char *txt)

Set the value of a cell.

> **Nota**
>
> New roes/columns are added automatically if required

Parametri

- **obj** -- pointer to a Table object
- **row** -- id of the row [0 .. row_cnt -1]
- **col** -- id of the column [0 .. col_cnt -1]
- **txt** -- text to display in the cell. It will be copied and saved so this variable is not required after this function call.

void **lv_table_set_cell_value_fmt**(lv_obj_t *obj, uint32_t row, uint32_t col, const char *fmt, ...)

Set the value of a cell. Memory will be allocated to store the text by the table.

> **Nota**
>
> New roes/columns are added automatically if required

Parametri

- **obj** -- pointer to a Table object
- **row** -- id of the row [0 .. row_cnt -1]
- **col** -- id of the column [0 .. col_cnt -1]
- **fmt** -- `printf`-like format

void **lv_table_set_row_count**(lv_obj_t *obj, uint32_t row_cnt)

Set the number of rows

Parametri

- **obj** -- table pointer to a Table object
- **row_cnt** -- number of rows

void **lv_table_set_column_count**(lv_obj_t *obj, uint32_t col_cnt)

Set the number of columns

Parametri

- **obj** -- table pointer to a Table object
- **col_cnt** -- number of columns.

void **lv_table_set_column_width**(lv_obj_t *obj, uint32_t col_id, int32_t w)

Set the width of a column

Parametri

- **obj** -- table pointer to a Table object
- **col_id** -- id of the column [0 .. LV_TABLE_COL_MAX -1]

- **w** -- width of the column

void **lv_table_set_cell_ctrl**(lv_obj_t *obj, uint32_t row, uint32_t col, *lv_table_cell_ctrl_t* ctrl)

Add control bits to the cell.

Parametri

- **obj** -- pointer to a Table object
- **row** -- id of the row [0 .. row_cnt -1]
- **col** -- id of the column [0 .. col_cnt -1]
- **ctrl** -- OR-ed values from ::lv_table_cell_ctrl_t

void **lv_table_clear_cell_ctrl**(lv_obj_t *obj, uint32_t row, uint32_t col, *lv_table_cell_ctrl_t* ctrl)

Clear control bits of the cell.

Parametri

- **obj** -- pointer to a Table object
- **row** -- id of the row [0 .. row_cnt -1]
- **col** -- id of the column [0 .. col_cnt -1]
- **ctrl** -- OR-ed values from ::lv_table_cell_ctrl_t

void **lv_table_set_cell_user_data**(lv_obj_t *obj, uint16_t row, uint16_t col, void *user_data)

Add custom user data to the cell.

Parametri

- **obj** -- pointer to a Table object
- **row** -- id of the row [0 .. row_cnt -1]
- **col** -- id of the column [0 .. col_cnt -1]
- **user_data** -- pointer to the new user_data. Should be allocated by `lv_malloc`, and it will be freed automatically when the table is deleted or when the cell is dropped due to lower row or column count.

void **lv_table_set_selected_cell**(lv_obj_t *obj, uint16_t row, uint16_t col)

Set the selected cell

Parametri

- **obj** -- pointer to a table object
- **row** -- id of the cell row to select
- **col** -- id of the cell column to select

const char ***lv_table_get_cell_value**(lv_obj_t *obj, uint32_t row, uint32_t col)

Get the value of a cell.

Parametri

- **obj** -- pointer to a Table object
- **row** -- id of the row [0 .. row_cnt -1]
- **col** -- id of the column [0 .. col_cnt -1]

Ritorna

text in the cell

uint32_t **lv_table_get_row_count**(lv_obj_t *obj)

Get the number of rows.

Parametri

obj -- table pointer to a Table object

Ritorna

number of rows.

uint32_t **lv_table_get_column_count**(lv_obj_t *obj)

Get the number of columns.

Parametri

obj -- table pointer to a Table object

Ritorna

number of columns.

int32_t **lv_table_get_column_width**(lv_obj_t *obj, uint32_t col)

Get the width of a column

Parametri
- **obj** -- table pointer to a Table object
- **col** -- id of the column [0 .. LV_TABLE_COL_MAX -1]

Ritorna
width of the column

bool **lv_table_has_cell_ctrl**(lv_obj_t *obj, uint32_t row, uint32_t col, *lv_table_cell_ctrl_t* ctrl)

Get whether a cell has the control bits

Parametri
- **obj** -- pointer to a Table object
- **row** -- id of the row [0 .. row_cnt -1]
- **col** -- id of the column [0 .. col_cnt -1]
- **ctrl** -- OR-ed values from ::lv_table_cell_ctrl_t

Ritorna
true: all control bits are set; false: not all control bits are set

void **lv_table_get_selected_cell**(lv_obj_t *obj, uint32_t *row, uint32_t *col)

Get the selected cell (pressed and or focused)

Parametri
- **obj** -- pointer to a table object
- **row** -- pointer to variable to store the selected row (LV_TABLE_CELL_NONE: if no cell selected)
- **col** -- pointer to variable to store the selected column (LV_TABLE_CELL_NONE: if no cell selected)

void ***lv_table_get_cell_user_data**(lv_obj_t *obj, uint16_t row, uint16_t col)

Get custom user data to the cell.

Parametri
- **obj** -- pointer to a Table object
- **row** -- id of the row [0 .. row_cnt -1]
- **col** -- id of the column [0 .. col_cnt -1]

Variables

const lv_obj_class_t **lv_table_class**

lv_table_private.h

struct **_lv_table_cell_t**

#include <lv_table_private.h> Cell data

Public Members

lv_table_cell_ctrl_t **ctrl**

void ***user_data**

Custom user data

char **txt**[1]

Variable length array

struct **_lv_table_t**

#include <lv_table_private.h> Table data

Public Members

lv_obj_t **obj**

uint32_t **col_cnt**

uint32_t **row_cnt**

lv_table_cell_t ****cell_data**

int32_t ***row_h**

int32_t ***col_w**

uint32_t **col_act**

uint32_t **row_act**

14.24.33 tabview

lv_tabview.h

Functions

lv_obj_t ***lv_tabview_create**(lv_obj_t *parent)

Create a tabview widget

Parametri

parent -- pointer to a parent widget

Ritorna

the created tabview

lv_obj_t ***lv_tabview_add_tab**(lv_obj_t *obj, const char *name)

Add a tab to the tabview

Parametri

- **obj** -- pointer to a tabview widget
- **name** -- the name of the tab, it will be displayed on the tab bar

Ritorna

the widget where the content of the tab can be created

void **lv_tabview_rename_tab**(lv_obj_t *obj, uint32_t idx, const char *new_name)

Change the name of the tab

Parametri

- **obj** -- pointer to a tabview widget
- **idx** -- the index of the tab to rename
- **new_name** -- the new name as a string

void **lv_tabview_set_active**(lv_obj_t *obj, uint32_t idx, *lv_anim_enable_t* anim_en)

Show a tab

Parametri

- **obj** -- pointer to a tabview widget
- **idx** -- the index of the tab to show
- **anim_en** -- LV_ANIM_ON/OFF

void **lv_tabview_set_tab_bar_position**(lv_obj_t *obj, *lv_dir_t* dir)

Set the position of the tab bar

Parametri

- **obj** -- pointer to a tabview widget
- **dir** -- LV_DIR_TOP/BOTTOM/LEFT/RIGHT

void **lv_tabview_set_tab_bar_size**(lv_obj_t *obj, int32_t size)

Set the width or height of the tab bar

Parametri

- **obj** -- pointer to tabview widget
- **size** -- size of the tab bar in pixels or percentage. will be used as width or height based on the position of the tab bar)

uint32_t **lv_tabview_get_tab_count**(lv_obj_t *obj)

Get the number of tabs

Parametri

obj -- pointer to a tabview widget

Ritorna

the number of tabs

uint32_t **lv_tabview_get_tab_active**(lv_obj_t *obj)

Get the current tab's index

Parametri

obj -- pointer to a tabview widget

Ritorna

the zero based index of the current tab

lv_obj_t ***lv_tabview_get_content**(lv_obj_t *obj)

Get the widget where the container of each tab is created

Parametri

obj -- pointer to a tabview widget

Ritorna

the main container widget

lv_obj_t ***lv_tabview_get_tab_bar**(lv_obj_t *obj)

Get the tab bar where the buttons are created

Parametri

obj -- pointer to a tabview widget

Ritorna

the tab bar

Variables

const lv_obj_class_t **lv_tabview_class**

lv_tabview_private.h

struct **_lv_tabview_t**

Public Members

lv_obj_t **obj**

uint32_t **tab_cur**

lv_dir_t **tab_pos**

14.24.34 textarea

lv_textarea.h

Defines

LV_TEXTAREA_CURSOR_LAST

LV_PART_TEXTAREA_PLACEHOLDER

Enums

enum **_lv_property_textarea_id_t**

Values:

enumerator **LV_PROPERTY_TEXTAREA_TEXT**

enumerator **LV_PROPERTY_TEXTAREA_PLACEHOLDER_TEXT**

enumerator **LV_PROPERTY_TEXTAREA_CURSOR_POS**

enumerator **LV_PROPERTY_TEXTAREA_CURSOR_CLICK_POS**

enumerator **LV_PROPERTY_TEXTAREA_PASSWORD_MODE**

enumerator **LV_PROPERTY_TEXTAREA_PASSWORD_BULLET**

enumerator **LV_PROPERTY_TEXTAREA_ONE_LINE**

enumerator **LV_PROPERTY_TEXTAREA_ACCEPTED_CHARS**

enumerator **LV_PROPERTY_TEXTAREA_MAX_LENGTH**

enumerator **LV_PROPERTY_TEXTAREA_INSERT_REPLACE**

enumerator **LV_PROPERTY_TEXTAREA_TEXT_SELECTION**

enumerator **LV_PROPERTY_TEXTAREA_PASSWORD_SHOW_TIME**

enumerator **LV_PROPERTY_TEXTAREA_LABEL**

enumerator **LV_PROPERTY_TEXTAREA_TEXT_IS_SELECTED**

enumerator **LV_PROPERTY_TEXTAREA_CURRENT_CHAR**

enumerator **LV_PROPERTY_TEXTAREA_END**

Functions

lv_obj_t ***lv_textarea_create**(lv_obj_t *parent)

Create a text area object

Parametri

parent -- pointer to an object, it will be the parent of the new text area

Ritorna

pointer to the created text area

void **lv_textarea_add_char**(lv_obj_t *obj, uint32_t c)

Insert a character to the current cursor position. To add a wide char, e.g. 'Á' use lv_text_encoded_conv_wc('Á')

Parametri

- **obj** -- pointer to a text area object
- **c** -- a character (e.g. 'a)

void **lv_textarea_add_text**(lv_obj_t *obj, const char *txt)

Insert a text to the current cursor position

Parametri

- **obj** -- pointer to a text area object
- **txt** -- a '\0' terminated string to insert

void **lv_textarea_delete_char**(lv_obj_t *obj)

Delete a the left character from the current cursor position

Parametri

obj -- pointer to a text area object

void **lv_textarea_delete_char_forward**(lv_obj_t *obj)

Delete the right character from the current cursor position

Parametri

obj -- pointer to a text area object

void **lv_textarea_set_text**(lv_obj_t *obj, const char *txt)

Set the text of a text area

Parametri

- **obj** -- pointer to a text area object
- **txt** -- pointer to the text

void **lv_textarea_set_placeholder_text**(lv_obj_t *obj, const char *txt)

Set the placeholder text of a text area

Parametri

- **obj** -- pointer to a text area object
- **txt** -- pointer to the text

void **lv_textarea_set_cursor_pos**(lv_obj_t *obj, int32_t pos)

Set the cursor position

Parametri

- **obj** -- pointer to a text area object
- **pos** -- the new cursor position in character index < 0 : index from the end of the text LV_TEXTAREA_CURSOR_LAST: go after the last character

void **lv_textarea_set_cursor_click_pos**(lv_obj_t *obj, bool en)

Enable/Disable the positioning of the cursor by clicking the text on the text area.

Parametri

- **obj** -- pointer to a text area object
- **en** -- true: enable click positions; false: disable

void **lv_textarea_set_password_mode**(lv_obj_t *obj, bool en)

Enable/Disable password mode

Parametri

- **obj** -- pointer to a text area object
- **en** -- true: enable, false: disable

void **lv_textarea_set_password_bullet**(lv_obj_t *obj, const char *bullet)

Set the replacement characters to show in password mode

Parametri

- **obj** -- pointer to a text area object
- **bullet** -- pointer to the replacement text

void **lv_textarea_set_one_line**(lv_obj_t *obj, bool en)

Configure the text area to one line or back to normal

Parametri

- **obj** -- pointer to a text area object
- **en** -- true: one line, false: normal

void **lv_textarea_set_accepted_chars**(lv_obj_t *obj, const char *list)

Set a list of characters. Only these characters will be accepted by the text area

Parametri

- **obj** -- pointer to a text area object
- **list** -- list of characters. Only the pointer is saved. E.g. "+-.,0123456789"

void **lv_textarea_set_max_length**(lv_obj_t *obj, uint32_t num)

Set max length of a Text Area.

Parametri

- **obj** -- pointer to a text area object
- **num** -- the maximal number of characters can be added (`lv_textarea_set_text` ignores it)

void **lv_textarea_set_insert_replace**(lv_obj_t *obj, const char *txt)

In `LV_EVENT_INSERT` the text which planned to be inserted can be replaced by another text. It can be used to add automatic formatting to the text area.

Parametri

- **obj** -- pointer to a text area object
- **txt** -- pointer to a new string to insert. If `""` no text will be added. The variable must be live after the `event_cb` exists. (Should be `global` or `static`)

void **lv_textarea_set_text_selection**(lv_obj_t *obj, bool en)

Enable/disable selection mode.

Parametri

- **obj** -- pointer to a text area object
- **en** -- true or false to enable/disable selection mode

void **lv_textarea_set_password_show_time**(lv_obj_t *obj, uint32_t time)

Set how long show the password before changing it to '*'

Parametri

- **obj** -- pointer to a text area object
- **time** -- show time in milliseconds. 0: hide immediately.

void **lv_textarea_set_align**(lv_obj_t *obj, *lv_text_align_t* align)

Deprecated:

Use the normal text_align style property instead Set the label's alignment. It sets where the label is aligned (in one line mode it can be smaller than the text area) and how the lines of the area align in case of multiline text area

Parametri

- **obj** -- pointer to a text area object
- **align** -- the align mode from *lv_text_align_t*

const char ***lv_textarea_get_text**(const lv_obj_t *obj)

Get the text of a text area. In password mode it gives the real text (not '*'s).

Parametri

obj -- pointer to a text area object

Ritorna
pointer to the text

const char *`lv_textarea_get_placeholder_text`(lv_obj_t *obj)

Get the placeholder text of a text area

Parametri
obj -- pointer to a text area object

Ritorna
pointer to the text

lv_obj_t *`lv_textarea_get_label`(const lv_obj_t *obj)

Get the label of a text area

Parametri
obj -- pointer to a text area object

Ritorna
pointer to the label object

uint32_t `lv_textarea_get_cursor_pos`(const lv_obj_t *obj)

Get the current cursor position in character index

Parametri
obj -- pointer to a text area object

Ritorna
the cursor position

bool `lv_textarea_get_cursor_click_pos`(lv_obj_t *obj)

Get whether the cursor click positioning is enabled or not.

Parametri
obj -- pointer to a text area object

Ritorna
true: enable click positions; false: disable

bool `lv_textarea_get_password_mode`(const lv_obj_t *obj)

Get the password mode attribute

Parametri
obj -- pointer to a text area object

Ritorna
true: password mode is enabled, false: disabled

const char *`lv_textarea_get_password_bullet`(lv_obj_t *obj)

Get the replacement characters to show in password mode

Parametri
obj -- pointer to a text area object

Ritorna
pointer to the replacement text

bool `lv_textarea_get_one_line`(const lv_obj_t *obj)

Get the one line configuration attribute

Parametri
obj -- pointer to a text area object

Ritorna
true: one line configuration is enabled, false: disabled

const char *`lv_textarea_get_accepted_chars`(lv_obj_t *obj)

Get a list of accepted characters.

Parametri
obj -- pointer to a text area object

Ritorna
list of accented characters.

uint32_t `lv_textarea_get_max_length`(lv_obj_t *obj)

Get max length of a Text Area.

Parametri
obj -- pointer to a text area object
Ritorna
the maximal number of characters to be add

bool **lv_textarea_text_is_selected**(const lv_obj_t *obj)

Find whether text is selected or not.
Parametri
obj -- pointer to a text area object
Ritorna
whether text is selected or not

bool **lv_textarea_get_text_selection**(lv_obj_t *obj)

Find whether selection mode is enabled.
Parametri
obj -- pointer to a text area object
Ritorna
true: selection mode is enabled, false: disabled

uint32_t **lv_textarea_get_password_show_time**(lv_obj_t *obj)

Set how long show the password before changing it to '*'
Parametri
obj -- pointer to a text area object
Ritorna
show time in milliseconds. 0: hide immediately.

uint32_t **lv_textarea_get_current_char**(lv_obj_t *obj)

Get a the character from the current cursor position
Parametri
obj -- pointer to a text area object
Ritorna
a the character or 0

void **lv_textarea_clear_selection**(lv_obj_t *obj)

Clear the selection on the text area.
Parametri
obj -- pointer to a text area object

void **lv_textarea_cursor_right**(lv_obj_t *obj)

Move the cursor one character right
Parametri
obj -- pointer to a text area object

void **lv_textarea_cursor_left**(lv_obj_t *obj)

Move the cursor one character left
Parametri
obj -- pointer to a text area object

void **lv_textarea_cursor_down**(lv_obj_t *obj)

Move the cursor one line down
Parametri
obj -- pointer to a text area object

void **lv_textarea_cursor_up**(lv_obj_t *obj)

Move the cursor one line up
Parametri
obj -- pointer to a text area object

Variables

const lv_obj_class_t **lv_textarea_class**

lv_textarea_private.h

struct **_lv_textarea_t**

#include <lv_textarea_private.h> Data of text area

Public Members

lv_obj_t **obj**

lv_obj_t ***label**

Label of the text area

char ***placeholder_txt**

Place holder label. only visible if text is an empty string

char ***pwd_tmp**

Used to store the original text in password mode

char ***pwd_bullet**

Replacement characters displayed in password mode

const char ***accepted_chars**

Only these characters will be accepted. NULL: accept all

uint32_t **max_length**

The max. number of characters. 0: no limit

uint32_t **pwd_show_time**

Time to show characters in password mode before change them to '*'

int32_t **valid_x**

Used when stepping up/down to a shorter line. (Used by the library)

uint32_t **pos**

The current cursor position (0: before 1st letter; 1: before 2nd letter ...)

lv_area_t **area**

Cursor area relative to the Text Area

uint32_t **txt_byte_pos**

Byte index of the letter after (on) the cursor

uint8_t **show**

Cursor is visible now or not (Handled by the library)

uint8_t **click_pos**

1: Enable positioning the cursor by clicking the text area

struct *_lv_textarea_t* **cursor**

uint32_t **sel_start**

Temporary values for text selection

uint32_t **sel_end**

uint8_t **text_sel_in_prog**

User is in process of selecting

uint8_t **text_sel_en**

Text can be selected on this text area

uint8_t **pwd_mode**

Replace characters with '*'

uint8_t **one_line**

One line mode (ignore line breaks)

14.24.35 tileview

lv_tileview.h

Functions

lv_obj_t ***lv_tileview_create**(lv_obj_t *parent)

Create a tileview object

Parametri

parent -- pointer to an object, it will be the parent of the new tileview

Ritorna

pointer to the created tileview

lv_obj_t ***lv_tileview_add_tile**(lv_obj_t *tv, uint8_t col_id, uint8_t row_id, *lv_dir_t* dir)

Add a tile to the tileview

Parametri

- **tv** -- pointer to the tileview object
- **col_id** -- column id of the tile
- **row_id** -- row id of the tile
- **dir** -- direction to move to the next tile

Ritorna

pointer to the added tile object

void **lv_tileview_set_tile**(lv_obj_t *tv, lv_obj_t *tile_obj, *lv_anim_enable_t* anim_en)

Set the active tile in the tileview.

Parametri

- **parent** -- pointer to the tileview object
- **tile_obj** -- pointer to the tile object to be set as active
- **anim_en** -- animation enable flag (LV_ANIM_ON or LV_ANIM_OFF)

void **lv_tileview_set_tile_by_index**(lv_obj_t *tv, uint32_t col_id, uint32_t row_id, *lv_anim_enable_t* anim_en)

Set the active tile by index in the tileview

Parametri

- **tv** -- pointer to the tileview object
- **col_id** -- column id of the tile to be set as active
- **row_id** -- row id of the tile to be set as active
- **anim_en** -- animation enable flag (LV_ANIM_ON or LV_ANIM_OFF)

lv_obj_t ***lv_tileview_get_tile_active**(lv_obj_t *obj)

Get the currently active tile in the tileview

Parametri

obj -- pointer to the tileview object

Ritorna

pointer to the currently active tile object

Variables

const lv_obj_class_t **lv_tileview_class**

const lv_obj_class_t **lv_tileview_tile_class**

lv_tileview_private.h

struct **_lv_tileview_t**

Public Members

lv_obj_t **obj**

lv_obj_t ***tile_act**

struct **_lv_tileview_tile_t**

Public Members

lv_obj_t **obj**

lv_dir_t **dir**

14.24.36 win

lv_win.h

Functions

lv_obj_t ***lv_win_create**(lv_obj_t *parent)

Create a window widget

Parametri

parent -- pointer to a parent widget

Ritorna

the created window

lv_obj_t ***lv_win_add_title**(lv_obj_t *win, const char *txt)

Add a title to the window

Parametri

- **obj** -- pointer to a window widget

- **txt** -- the text of the title

Ritorna

the widget where the content of the title can be created

lv_obj_t ***lv_win_add_button**(lv_obj_t *win, const void *icon, int32_t btn_w)

Add a button to the window

Parametri

- **obj** -- pointer to a window widget
- **icon** -- an icon to be displayed on the button
- **btn_w** -- width of the button

Ritorna

the widget where the content of the button can be created

lv_obj_t ***lv_win_get_header**(lv_obj_t *win)

Get the header of the window

Parametri

win -- pointer to a window widget

Ritorna

the header of the window

lv_obj_t ***lv_win_get_content**(lv_obj_t *win)

Get the content of the window

Parametri

win -- pointer to a window widget

Ritorna

the content of the window

Variables

const lv_obj_class_t **lv_win_class**

lv_win_private.h

struct **_lv_win_t**

Public Members

lv_obj_t **obj**