LittlevGL documentation (Brazilian Portuguese)

Table of contents

Bem vindo

Escrito para v5.3, revisão 2

Bem vindo a Biblioteca de interface gráfica LittlevGL

LittlevGL é um software gráfico aberto com bibliotecas que providencia tudo que você precisa para criar um GUI (Interface Gráfica de Usuário), com elementoos fáceis de usar e efeitos visuais bonitos e usando pouca memória

Principais características

- **Poderosos blocos de construção** botões, gráficos, listas, controles deslizantes, imagens, etc.
- **Gráficos avançados** com animações, anti-aliasing, opacidade, deslizamento suave
- **Vários dispositivos de entrada** touch pad, mouse, teclado, codificadores, botões, etc.
- **Suporte à multi-idioma** com codificação UTF-8
- **Elementos gráficos totalmente customizáveis**
- **Hardware independente** para usar com qualquer microcontrolador ou display
- **Escalável** para operar com pequena quantidade de memória (50 KB de Flash, 10 KB de RAM)
- **OS, GPU e memória externa supportados** (opcional)
- **Único frame buffer** operações até mesmo com avanço de efeitos gráficos
- **Escrito em C** para máxima compatibilidade
- **Simulador** para desenvolver no PC sem necessidade de um hardware específico
- **Tutoriais, examplos, temas** para desenvolvimento rápido

Como iniciar?

Leia a documentação

É sempre uma boa idéia ler a documentação primeiro. Não se apresse. Tome algum tempo para aprender o básico.

Você pode iniciar aqui Portabilidade ou com a Introdução para essa biblioteca.

Tutorial

Há um tutorial para mostrar para você as partes mais importantes passo a passo.

Está localizado no repositório lb_examples

Teste o LittlevGL no simulador PC

Se você ainda não possui um hardware com display você pode testar a biblioteca no seu PC. O simulador PC usa uma janela no seu monitor para simular um display físico e usa seu mouse para clicar naquele display.

O simulador funciona com Windows, Linux e também no OSX.

Aqui você pode aprender como configurar o simulador: Simulador para PC

Contribuição

Use o rastreador de problemas (issue tracker) do GitHub para:

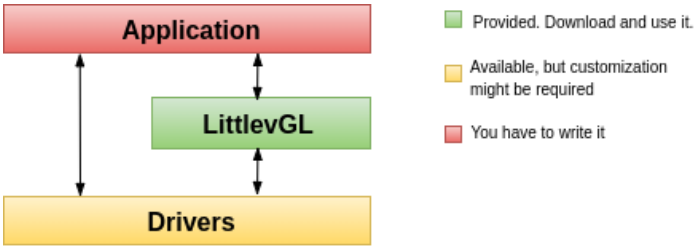
- Reportar bugs
- Sugerir novos recursos
- Adicionar novos recursos
- Ajudar os outros

Antes de contribuir, leia o documento relacionado: CONTRIBUINDO

Portabilidade

Escrito para v5.2

Arquitetura do sistema



Aplicação

Sua aplicação na qual cria o GUI e controlador (handle) para a tarefa específica.

LittlevGL

Os próprios gráficos da biblioteca. Sua aplicação pode comunicar com a biblioteca para criar a GUI. Ele contém uma interface HAL (Camada Abstrata de Hardware) para registrar seu display e drivers de entrada de dispositivos.

Driver

Junto ao seu driver específico, a biblioteca contém funções para controlar seu display, opcionalmente uma GPU e leitura do touchpad ou botões.

Existem **duas configurações típicas para hardware** dependendo da MCU que pussui um driver periférico LCD/TFT ou não. Em ambos os casoos, um frame buffer será necessário para armazenar a imagem atual na tela.

MCU com driver TFT/LCD

Se sua MCU tem um driver TFT/LCD então você pode conectar um display diretamente via interface RGB. Neste caso, o frame buffer pode estar na sia RAM interna (se o MCU possui RAM suficiente) ou na RAM externa (se a MCU possui uma interface de memória).

Controlador de display externo

Se o MCU não possui um driver TFT/LCD então contrlador de display externo (I.e. SSD1963, SSD1306, ILI9341) precisa ser usado. Neste caso, o MCU pode se comunicar com o contralador do display via porta paralela, SPI ou algumas vezes I2C. O frame buffer é usualmente localizado no controlador de display na qual economiza muita RAM para a MCU.

Requerimentos

- **Microcontrolador ou processador de 16, 32 ou 64 bit**
- **Velocidade de clock de 16 MHz**
- **8 KB de RAM** para dados estáticos e **>2 KB de RAM** para dados dinâmicos (objetos gráficos)
- **64 KB de memória permanente** (flash)
- **Opcionalmente ~1/10 do tamanho da tela de memória** para buffer interno (em 240 x 320, com cores de 16 bit é de aprox. 15 KB)

O LittlevGL é projetado para ser altamente portavel e não necessitar de nenhum **recurso externo**:

- Nenhuma RAM externa é requerida (mas suportado)
- Nenhum número de vírgula flutuante são usados
- Nenhuma GPU é necessária (mas suportado)
- Somente um único frame buffer é requerido e localizado em:
  - RAM interna ou

- RAM externa ou
- Controlador da memória do display externo

Se você quiser **reduzir** o **recurso de hardware** requerido você pode:

- Desativar tipos de objetos não usados para economizar RAM e ROM
- Mudar o tamanho do buffer gráfico para economizar RAM
- Usar estilos mais simples para reduzir o tempo de renderização

Preparação do projeto

Obtendo a biblioteca

O Littlev Graphics Library está disponível no GitHub: https://github.com/littlevgl/lvgl. Você pode clonar ou fazer download da última versão da biblioteca daqui ou você pode usar a página de Download também.

As bibliotecas gráficas está no diretório **lvgl** na qual deve ser copiado dentro do seu projeto.

Arquivo de configuração

Existe um arquivo cabeçalho de configuração para o LittlevGL: **lv_conf.h**. Ele configura o comportamento básico da biblioteca durante o tempo de compilação. Desative os módulos não usados, configurações e ajuste do tamanho da memória de buffers, etc.

Copie _lvgl/lv_conf_templ.h_ próximo ao diretório *lvgl* e renomeie-o para _lv_conf.h_. Abra o arquivo e apague o primeiro *#if* e o último *#endif* para ativar o conteúdo. No arquivo de configuração, comentários explicam o siginificado das opções. Verifique pelo menos essas três opções e modifique-as de acordo com seu hardware.

1. **LV_HOR_RES** Resolução horizontal de seu display
2. **LV_VER_RES** Resolução vertical de seu display
3. **LV_COLOR_DEPTH** 8 para (RG332), 16 para (RGB565) ou 32 para (RGB888 and ARGB8888).

Inicialização

Para usar a biblioteca você tem que inicializá-la além de outros componentes também. A ordem da inicialização é:

1. Chamar _lv_init()_
2. Inicializar seus drivers
3. Registrar o display e a entrada de dispositivos dos drives dentro do LittlevGL. (Veja abaixo)

Portando a biblioteca

Para colocar o LittlevGL dentro do seu projeto, primeiramente você tem que providenciar algumas funções e registrá-la na biblioteca de gráficos.

Interface do display

Para configurar um display uma variável do tipo **lv_disp_drv_t** deve ser inicializada:

```
lv_disp_drv_t disp_drv;
lv_disp_drv_init(&disp_drv); /*Inicialização básica*/
disp_drv. ...=...; /*Inicializar os campos aqui. Veja abaixo.*/
lv_disp_drv_register(&disp_drv); /*Registrar o driver dentro do LittlevGL*/
```

Você pode configurar o driver para diferentes modos de operação. Para aprender mais sobre modos de desenho, visite Desenhando e renderizando.

Bufferização interna (VDB)

A biblioteca de gráficos funciona com mecanismo de buffer para criar características de avanços gráficos com somente um frame buffer. Um

buffer interno é chamado de VDB (Display Virtual de Buffer) a seu tamanho pode ser ajustado dentro do lv_conf.h com _LV_VDB_SIZE_. quando _LV_VDB_SIZE_ > 0 então a bufferização interna é usada e você tem que providenciar uma função na qual esvazie o conteúdo desse buffer para o seu display:

```
disp_drv.disp_flush = my_disp_flush;
...
void my_disp_flush(int32_t x1, int32_t y1, int32_t x2, int32_t y2, const lv_color_t* color_p)
{
    /*TODO Copie 'color_p' para a área específica*/
    /*Chame 'lv_flush_ready()' quando estiver pronto*/lv_flush_ready();
}
```

Na função flush, você pode usar DMA ou qualquer hardware para fazer o despejo para o background, mas quando o despejo estiver pronto você tem que chamar

```
lv_flush_ready();
```

Aceleração de Hardware (GPU)

Primeiramente, usar GPU é totalmente opcional. Mas se sua MCU suporta aceleração gráfica então você pode usá-la. Os campos _mem_blend_ e _mem_fill_ do driver do display é usado para interfacear com a GPU. As funções relacionadas à GPU pode ser usada somente se o buffer interno (VDB) estiver desativada.

```
disp_drv.mem_blend = my_mem_blend;     /*Mistura os dois vetores de cores usando opacidade*/
disp_drv.mem_fill = my_mem_fill;       /*Preenche um vetor com uma cor*/
...
void my_mem_blend(lv_color_t* dest, constlv_color_t* src, uint32_t length, lv_opa_t opa)
{
  /*TODO Copie 'src' para 'dest' mas misture isso com 'opa' alpha*/
}

void my_mem_fill(lv_color_t* dest, uint32_t length, lv_color_t color)
{
  /*TODO Preencha os pixels 'length' dentro do 'dest' com 'color'*/
}
```

Tirando o buffer do desenho

É possível desenhar diretamente para um frame buffer quando o buffer interno está desativado (LV_VDB_SIZE = 0).

```
disp_drv.disp_fill = my_disp_fill;/*Preenche uma área no frame buffer*/
disp_drv.disp_map = my_disp_map;/*Copia uma color_map (e.g. imagem) dentro do frame buffer*/
...
void my_disp_map(int32_t x1,int32_t y1,int32_t x2,int32_t y2,constlv_color_t* color_p)
{
  /*TODO Copia 'color_p' para uma área específica*/
}

void my_disp_fill(int32_t x1,int32_t y1,int32_t x2,int32_t y2,lv_color_t color)
{
  /*TODO Preenche a área especificada com 'color'*/
}
```

Mantenha em mente que desse jeito durante o 'refresh' alguns artefatos pode estar visível porque as camadas estão desenhadas entre elas. E alguns gráficos de características alto nível como anti-aliasing, opacidade ou sombras não estão disponíveis nesta configuração.

Se você usa um controlador de display externo na qual suporta preenchimento acelerado (e.g. RA8876) então você pode usar este recurso no _disp_fill()_

Interface de dispotitivo de entrada

Para configurar um dispositivo de entrada uma variável **lv_indev_drv_t** precisa ser inicializada:

```
lv_indev_drv_t indev_drv;lv_indev_drv_init(&indev_drv);  /*Inicialização básica*/
indev_drv.type =.../*Veja abaixo.*/
indev_drv.read =.../*Veja abaixo.*/
lv_indev_drv_register(&indev_drv);  /*Registra o driver no LittlevGL*/
```

type pode ser

- LV_INDEV_TYPE_POINTER: touchpad ou mouse

- LV_INDEV_TYPE_KEYPAD: teclado
- LV_INDEV_TYPE_ENCODER: esquerda, direita, apertar
- LV_INDEV_TYPE_BUTTON: botão externo pressionando a tela

read é um função ponteiro na qual será chamada periodicamente para reportar o estado corrente de um dispositivo de entrada. Ele pode também dar um buffer de dados e retornar *false* quando nenhum dados podem ser lidos ou *true* quando o buffer não está vazio.

To learn more about input devices visit Input devices. Para aprender mais sobre os dispositivos de entrada, visite Dispositivo de entrada.

Touchpad, mouse ou qualquer ponteiro

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

A leitura da função deve ser algo parecido como:

```
bool my_input_read(lv_indev_data_t*data)
{
    data->point.x = touchpad_x;
    data->point.y = touchpad_y;
    data->state = LV_INDEV_STATE_PR or LV_INDEV_STATE_REL;
    return false; /*Sem mais bufferização, sendo assim, sem mais dados para ler*/
}
```

> **NOTA IMPORTANTE:** Drivers de touchpad devem retornar os últimas coordenadas X/Y até mesmo quando o estado é `LV_INDEV_STATE_REL`.

Keypad ou teclado

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

A leitura da função deve ser algo parecido como:

```
bool keyboard_read(lv_indev_data_t*data){
  data->key = last_key();            /*Obtém o última tecla pressionada ou liberada*/
  if(key_pressed()){
    data->state = LV_INDEV_STATE_PR;
  }
  else{
    data->state = LV_INDEV_STATE_REL;
  }
  return false; /*Sem mais bufferização, sendo assim, sem mais dados para ler*/
}
```

Para usar um teclado:

- Registre uma função *read* (como acima) com o tipo _LV_INDEV_TYPE_KEYPAD_.
- O _USE_LV_GROUP_ tem que estar ativado dentro do _lv_conf.h_
- Um grupo de objeto tem que ser criado: _lv_group_create()_ e objetos tem que ser adicionado: _lv_group_add_obj()_
- O grupo criado tem que se assinado para um dispositivo de entrada: _lv_indev_set_group(my_indev, group1);_
- Use _LV_GROUP_KEY_..._ para navegar entre os objetos no grupo

Visite Touchpad-less navigation para aprender mais.

Codificador

Com um codificador você pode fazer 4 coisas:

1. pressionar seu botão
2. pressionar longamente seu botão
3. virar para esquerda

4. virar para direita

Ao ligar o codificador você pode focar no objeto próximo/anterior. Quando você pressiona o codificador em um simples objeto (como um botão). ele será clicado. Se você pressiona o codificador sobre um objeto complexo (como uma lista, caixa de mensagem, etc.) o objeto irá para o modo edição onde ligará o codificador, você pode navegar dentro do objeto. Para deixar o modo de edição pressione longamente o botão

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

A função de leitura deve se parecer como:

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

  return false; /*Sem bufferização então nennhum dado para ler*/
}
```

- Para usar um `ENCODER`, igualmente ao `KEYPA`, os objetos devem ser adicionados aos grupos.

Botão

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

A função de leitura deve se parecer como:

```
bool button_read(lv_indev_data_t*data){
    static uint32_t last_btn = 0;   /*Armazena o último botão pressionado*/
    int btn_pr = my_btn_read();     /*Obtém o ID (0,1,2...) do botão pressionado*/
    if(btn_pr >= 0) {               /*Existe botão pressionado?*/
       last_btn = btn_pr;           /*Salva o ID do botão pressionado*/
       data->state = LV_INDEV_STATE_PR;  /*Adiciona ao estado pressionado*/
    } else {
       data->state = LV_INDEV_STATE_REL; /*Adiciona ao estado liberado*/
    }

    data->btn = last_btn;            /*Adiciona o último botão*/

    return false;                    /*Sem bufferização então nennhum dado para ler*/
}
```

- Os botões precisam ser assinados para pixels na tela usando `lv_indev_set_button_points(indev, points_array)`. Onde _points_array_ se parece como `const lv_point_t points_array[] = { {12,30},{60,90}, ...}`

Interface tick

O LittlevGL usa um sistema tick. Chame a função `lv_tick_inc(tick_period)` periodicamente e chame o período de chamada em milisegundos. Por exemplo, se você chama em cada milisegundos: `lv_tick_inc(1)`

Carregamento de chamada

Para carregar as chamadas do LittlevGL você precisa chamar `lv_task_handler()` periodicamente em uma das seguintes:

- *while(1)* da função *main()*
- interrupção de temporizadores periodicos
- uma tarefa do OS periódica

A temporização não é critica mas deve se em torno de 5 milisegundos para manter o sistema responsível.

Exemplo:

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

O MCU pode ficar no estado **sleep** quando nenhuma entrada acontece. Nesse caso o main `while(1)` deve se parecer como:

```
while(1) {
  /*Operação normal em 1 seg*/
  if(lv_indev_get_inactive_time(NULL) < 1000) {
    lv_task_handler();
  }
  /*Dorme depois de 1 segundo inativo*/
  else {
    timer_stop();               /*Para o temporizador quando lv_tick_inc() é chamado*/
    sleep();            /*Faz o MCU dormir*/
  }
  my_delay_ms(5);
}
```

Você deve também adicionar essas linhas para seu dispositivo de entrara para ler a função quando um pressionamento acontece:

```
/*Força a tarefa de execução quando acorda*/
lv_tick_inc(LV_REFR_PERIOD);
timer_start();               /*Reinicia o temporizador quando lv_tick_inc() é chamado*/
lv_task_handler();
```

Em adição à `lv_indev_get_inactive_time()` você pode checar `lv_anim_count_running()` para ver se cada animação foi finalizada.

Usando com um sistema operacional

LittlevGL é **not thread-safe**. Além disso, ele é muito simples de usar dentro de um sistema operacional.

O **cenário simples** é não usar a tarefa do sistema operacional mas usar o `lv_task`s. Um _lv_task_ é uma fuunção chamada periodicamente em `lv_task_handler`. No _lv_task_ você pode obter os estados dos sensores, buffers etc e chamar as funções do LittlevGL para atualizar a GUI. Para criar um _lv_task_ use: `lv_task_create(my_func, period_ms, LV_TASK_PRIO_LOWEST/LOW/MID/HIGH/HIGHEST, custom_ptr)`

Se você precisa **usar outras tarefas ou threads** você precisa de um mutex na qual deve ser tomado antes de ser chamado `lv_task_handler` e lançado depois dele. Em adição, você terá que usar para esse mutex em outras tarefas e threads em volta de todo código relacionado ao LittlevGL (`lv_...`). Desse jeito você pode usar o LittlevGL em um ambiente real de multitarefa. Use um mutex para evitar chamadas concorrente das funções LittlevGL.

Exemplo de portabilidade

Aqui você encontrará um exemplo de codígo de portabilidade: Tutorial de portabilidada.

Simulador para PC

Escrito para v5.1

Você pode testar o Littlev Graphics Library **usando somente seu PC** sem qualquer placa de desenvolvimento. Escreva um código, rode no seu PC e veja o resultado no monitor. O software é de plataforma cruzada: São suportados Windows, Linux e OSX!

- Precisa somente de poucos minutos de configuração
- Custa $0. Sem custo para PCB e não precisa pagar por nenhum software
- Um display TFT é simulado e mostrado no monitor de seu PC
- O touch pad é substituido pelo seu mouse
- O código escrito é portável, você pode simplesmente copiar ele quando estiver usando um hardware embutido

Instalando o Eclipse CDT

Eclipse CDT é um IDE C/C++. Você pode usar outras IDEs também, mas neste tutorial é mostrado a configuração para o Eclipse CDT.

Eclipse é um software baseado em Java, certifique-se de que **Java Runtime Environment** está instalado no seu sistema.

Em distros baseado em pacotes Debian (ex.: Ubuntu): `sudo apt-get install default-jre`

Você pode fazer download do Eclipse CDT do https://eclipse.org/cdt/. Inicie o instalador e escolha *Eclipse CDT* da lista.

Instalando o SDL 2

O simulador PC usa a biblioteca de plataforma cruzada SDL 2 para simular um display TFT e o touch pad.

Linux

No **Linux** você pode facilmente instalar o SDL2 usando um terminal:

1. Encontre a versão atual do SDL2: `apt-cache search libsdl2 (Ex.: libsdl2-2.0-0)`
2. Instale o SDL2: `sudo apt-get install libsdl2-2.0-0` (troque com a versão encontrada)
3. Instale o pacote de desenvolvimento SDL2: `sudo apt-get install libsdl2-dev`
4. Se os pacotes essenciais não estão instalados ainda: `sudo apt-get install build-essential`

Windows

Se você está usando o **Windows** primeiramente você precisa instalar o MinGW (64 bit version). Depois disto siga os passoas para adicionar o SDL2:

1. Faça download das bibliotecas do SDL. Vá para https://www.libsdl.org/download-2.0.php e faça download do *Development Libraries: SDL2-devel-2.0.5-mingw.tar.gz*
2. Descompacte o arquivo e va para o diretório _x86_64-w64-mingw32_ (para MinGW 64 bit) ou para *i686-w64-mingw32* (para MinGW 32 bit)
3. Copie a pasta _...*mingw32/include/SDL2* para _C:/MinGW/.../x86_64-w64-mingw32/include_
4. Copie o conteúdo _...*mingw32/lib/* para _C:/MinGW/.../x86_64-w64-mingw32/lib_
5. Copie _...*mingw32/bin/SDL2.dll* para _{eclipse_worksapce}/pc_simulator/Debug/_. Faça isso depois do Eclipse ser instalado.

Nota: Se você usar o **Microsoft Visual Studio** ao invés do Eclipse então você não precisa instalar o MinGW

OSX

No **OSX** você pode facilmente instalar o SDL2 com brew: `brew install sdl2`

Se algo não está funcionando eu sugiro esse tutorial para iniciar com o SDL.

Projeto pré-configurado

Um projeto de biblioteca gráfica pré-configurada (baseada no último lançamento) está sempre disponível no projeto do simulador PC. Você pode

encontrá-lo no GitHub ou na página Download. O projeto está configurado para o Eclipse CDT.

Adicionar o projeto pré-configurado ao Eclipse CDT

Rode o Eclipse CDT. Ele mostrará um diálogo sobre o **atalho do espaço de trabalho*. Antes de aceitá-lo, cheque que o atalho e a cópia do download do projeto pré-configurado esteja lá. Agora você pode aceitar o atalho do espaço de trabalho. É claro que você pode modificar esse atalho mas no caso de copiar o projeto para aquela localização.

Feche a janela de início e vá para o **File->Import** e escolha **General->Existing project into Workspace**. **Browse the root directory** do seu projeto e clique **Finish**

No **Windows** você tem que fazer duas coias adicionais:

- Copie o **SDL2.dll** dentro da pasta Debug do seu proojeto
- Botão direito no progeto -> Project properties -> C/C++ Build -> Settings -> Libraries -> Add ... e adicione *mingw32* acima do SDLmain e do SDL. (A ordem é importante: mingw32, SDLmain, SDL)

Compilar e rodar

Agora que você está pronto, rode o Littlev Graphics Library no seu PC. Clique no Hammer Icon no topo da barra de menu para construir (Build) o projeto. Se você tiver feito tudo certo e não obtiver nenhum erro. Note que em alguns sistemas, alguns passos adicionais serão necessários para "ver" SDL 2 no Eclipse mas na maioria dos casos as configurações no projeto baixado é suficiente.

Depois do sucesso da compilação, clique no botão Play na barra de topo do menu para rodar o projeto. Agora a janela deve aparecer no meio da sua tela.

Agoora tudo está pronto para usar o Littlev Graphics Library na prática ou começar o desenvolvimento no seu PC.

Próximos passos

Para criar seu primeiro GUI LittlevGL você deve ler as páginas abaixo Portabilidade no conteúdo ao lado.

Objetos

Escrito para v5.1

No Littlev Graphics Library os **blocos básicos para construção** de uma interface de usuário são os objetos. Por exemplo:

- Botão
- Rótulo
- Imagem
- Lista
- Gráfico
- Área de texto

Clique para checar todos os tipos de Objetos existentes

Atributos de objetos

Atributos básicos

Os objetos tem um atributos básicos na qual são independentemente comum de seus tipos:

- Posição
- Tamanho
- Pai
- Desenho ativado
- Clique ativado, etc.

Você pode assinar/obter esses atributos com funções `lv_obj_set_...` e `lv_obj_get_...`. Por exemplo:

```
/*Atribui ao objeto atributos básicos*/
lv_obj_set_size(btn1, 100, 50);                     /*Tamanho do botão*/
lv_obj_set_pos(btn1, 20,30);                       /*Posição do botão*/
```

Para ver todas as funções disponíveis, visite os objetos de base da documentação

Atributos específicos

O tipos de objeto tem atributos especiais. Por exemplo um deslizador tem:

- Valores mínimos e máximos
- Valores atuais
- Função callback para novos valores
- Estilos

Para esses atributos, todos tipos de objetos tem uma única função API. Por exemplo para um deslizador:

```
/*Atribui ao deslizador os atributos específicos*/
lv_slider_set_range(slider1, 0, 100);           /*Atribui os valores mínimo e máximo*/
lv_slider_set_value(slider1, 40);            /*Atribui o valor (posição)*/
lv_slider_set_action(slider1, my_action);       /*Adiciona uma função callback*/
```

Mecanismo de trabalho dos objetos

Estrutura pai-filho

Um pai pode ser considerado como o container de seus filhos. Todos os objetos tem exatamente um objeto pai (exceto as telas), mas um pai pode ter ilimitado número de filhos. Não existe nenhuma limitação para o tipo de pai, mas há tipicamente objetos pai (ex.: botão) e filho (ex.:

rótulo).

Tela – o pai mais básico

A tela é um objeto especial na qual não possui um objeto pai. Sempre existe uma tela ativa. Por padrão, a biblioteca cria e carrega um. Para obter a tela ativa use a função `lv_scr_act()`.

Uma tela pode ser criada com qualquer tipo de objeto, por exemplo, um objeto básico ou uma imagem para fazer um papel de parede.

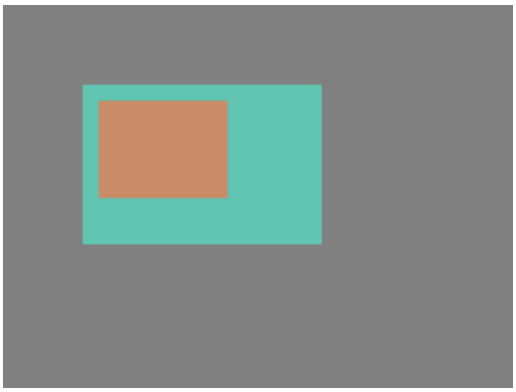
Movendo junto

Se a posição do pai é mudada, o filho irá mover junto com o pai. Contudo todas as posições são relativas ao pai. Para a coordenada (0;0), significa que os objetos permanecerão no topo ao lado esquerdo do pai, independentemente da posição do pai.

```
lv_obj_t * par = lv_obj_create(lv_scr_act(), NULL);     /*Cria um objeto pai na tela atual*/
lv_obj_set_size(par, 100, 80);                          /*Atribui um tamanho ao pai*/

lv_obj_t * obj1 = lv_obj_create(par, NULL);             /*Cria um objeto dentro do objeto pai criado anteriormente*/
lv_obj_set_pos(obj1, 10, 10);                       /*Atribui a posição do novo objeto*/
```

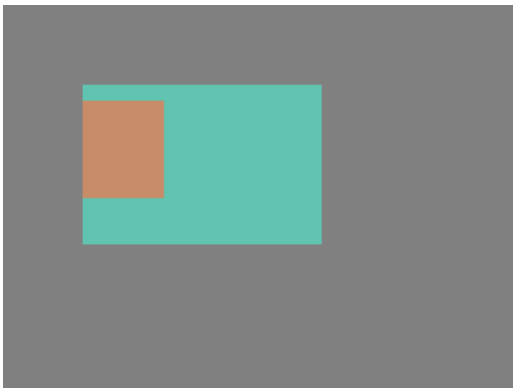
Modificar a posição do pai



```
lv_obj_set_pos(par, 50, 50);        /*Move o pai. O filho irá se mover junto com ele.*/
```

Visibilidade somente no pai

Se um filho parcialmente ou totalmente está fora de seu pai então as partes fora não será visível.



```
lv_obj_set_x(obj1, -30);        /*Move o filho um pequeno pedaço do pai*/
```

Criar - apagar objetos

Nos objetos de biblioteca gráfica podem somente ser criados ou apagados dinamicamente em tempo real. Significa que somente objetos criados consome RAM. Por exemplo, se voccê precisa de um gráfico, você pode criá-lo somente quando é requerido e apagá-lo depois de ter usado.

Todos os tipos de objetos tem sua própria função **criar** com um protótipo unificado. Ele precisa de dois parâmetros: um ponteiro do pai e opcionalmente um ponteiro para um outro objeto do mesmo tipo. Se o segundo parâmetro não é *NULL* então esses objetos será copiado para um novo. Para criar uma tela dê umm *NULL* como pai. O valor de retorno da função criada é um ponteiro ao objeto criado. Independentemente do tipo do objeto da variável comum **lv_obj_t** que será usado. Esse ponteiro pode ser usado mais tarde para atribuir ou obter atributos desse objeto. As funções de criação deve se parecer como essas:

```
lv_obj_t * lv_type_create(lv_obj_t * parent, lv_obj_t * copy);
```

Existe uma comum função **delete** para todos os tipos de objetos. Ela apaga o objeto e todos os seus filhos.

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

Você pode apagar somente os filhos de um objeto mas deixar o objeto pai "vivo":

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

Camadas

O objeto previamente criado (e seus filhos) serão desenhados anteriormente (mais perto do background). Em outras palavras, os objetos criados por último serão seus irmãos. Isto é muito importante, a ordem é calculada entre os objetos do mesmo nível ("irmãos").

Camadas podem ser adicionada facilmente criando 2 objetos (na qual podem ser transparentes) primeiramente 'A' e segundamente 'B'. 'A' e todos objetos sobre ele estarão no atrás (background_ e podem ser cobertos por 'B' e seus filhos.


```
/*Cria uma tela*/
lv_obj_t * scr = lv_obj_create(NULL, NULL);
lv_scr_load(scr);                                   /*Carrega a tela*/

/*Cria 2 botões*/
lv_obj_t * btn1 = lv_btn_create(scr, NULL);         /*Cria um botão na tela*/
lv_btn_set_fit(btn1, true, true);                   /*Ativa automaticamente o tamanho de acordo com o conteúdo*/
lv_obj_set_pos(btn1, 60, 40);                          /*Atribui a posição do botão*/

lv_obj_t * btn2 = lv_btn_create(scr, btn1);         /*Copia o primeiro botão*/
lv_obj_set_pos(btn2, 180, 80);                         /*Atribui a posição do botão*/

/*Adiciona rótulos aos botões*/
lv_obj_t * label1 = lv_label_create(btn1, NULL);    /*Cria um rótulo nos primeiros botões*/
lv_label_set_text(label1, "Button 1");              /*Atribui o texto ao rótulo*/

lv_obj_t * label2 = lv_label_create(btn2, NULL);      /*Cria um rótulo no segundo botão*/
lv_label_set_text(label2, "Button 2");                /*Atribui o texto ao rótulo*/

/*Apaga o segundo rótulo*/
lv_obj_del(label2);
```

Estilos

Escrito para v5.1

Para configurar a aparência dos estilos dos objetos que podem ser usados. Um estilo é uma estrutura variável com atributos como cores, preenchimento, visibilidade entre outros. Existe um estilo comum chamado: **lv_style_t**

Para atribuir os campos de uma estrutura `lv_style_t` você pode influenciar as aparências dos objetos usando aquele estilo.

Os objetos guardam somente um ponteiro para um estilo, então o estilo não pode ser uma variável local na qual é destruida depois que a função existe. **Você deve usar variáveis globais estáticas ou variáveis dinamicamente alocadas**

```
lv_style_t style_1;             /*OK! Variáveis globais para estilo são ótimas*/
static lv_style_t style_2;      /*OK! Variáveis estáticas fora das funções são ótimas*/
void my_screen_create(void)
{
  static lv_style_t style_3;    /*OK! Variáveis estáticas nas funções são ótimas*/
  lv_style_t style_1;           /*ERRADO! Estilos não podem ser variáveis locais*/

  ...
}
```

Propriedade de estilos

Um estilo tem 5 partes principais comuns: comum, corpo, texto, imagem e linha. Um objeto usará aqueles campos na qual são relevante para ele. Por exemplo, Linhas não se importa com letter_space. para ver quais campos são usados por um tipo de objeto veja suas documentações.

Os campos de uma estrutura de estilo são as seguintes:

- **Propriedades comuns**
  - **glass** 1: Não herda este estilo (veja abaixo)
- **Propriedades do estilo do corpo** Usado por um objeto tipo retângulo
  - **body.empty** Não preenche o retângulo (somente desenha a borda e/ou sombra)
  - **body.main_color** Cor principal (top color)
  - **body.grad_color** Cor do gradiente (cor de baixo)
  - **body.radius** Raio do canto. (configure o LV_RADIUS_CIRCLE para desenhar circulos)
  - **body.opa** Opacidade (0..255 ou LV_OPA_TRANSP, LV_OPA_10, LV_OPA_20 ... LV_OPA_COVER)
  - **body.border.color** Cor da borda
  - **body.border.width** Comprimento da borda
  - **body.border.part** Partes da borda (LV_BORDER_LEFT/RIGHT/TOP/BOTTOM/FULL ou valores 'OR'ed)
  - **body.border.opa** Opacidade da borda
  - **body.shadow.color** Cor da sombra
  - **body.shadow.width** Comprimento da sombra
  - **body.shadow.type** Tipo da sombra (LV_SHADOW_BOTTOM ou LV_SHADOW_FULL)
  - **body.padding.hor** Preenchimento horizontal
  - **body.padding.ver** Preenchimento Vertical
  - **body.padding.inner** Preenchimento interno
- **Propriedades de estilos de texto** Usado pelos objetos que mostrem textos
  - **text.color** Cor de texto
  - **text.font** Ponteiro para uma fonte

- **text.opa** Opacidade do texo (0..255 or LV_OPA_TRANSP, LV_OPA_10, LV_OPA_20 ... LV_OPA_COVER)
- **text.letter_space** Espaço da letra
- **text.line_space** Espaço da linha

- **Propriedades do estilo de imagem** Usado por objetos baseado a imagem ou icones sobre objetos
  - **image.color** Cor para imagem baseado em recolorização em brilho de pixels
  - **image.intense** Intensidade de recolorização (0..255 or LV_OPA_TRANSP, LV_OPA_10, LV_OPA_20 ... LV_OPA_COVER)
  - **image.opa** Opacidade da imagem (0..255 or LV_OPA_TRANSP, LV_OPA_10, LV_OPA_20 ... LV_OPA_COVER)
- **Propriedades do estilo de linha** Usado por objetos contendo linhas ou elementos baseado à linhas
  - **line.color** Cor da linha
  - **line.width** Comprimento da linha
  - **line.opa** Opacidade da linha (0..255 or LV_OPA_TRANSP, LV_OPA_10, LV_OPA_20 ... LV_OPA_COVER)

Usando estilos

Todo tipo de objeto tem uma única função para configurar estilo ou estilos.

Se o objeto tem somente um estilo - como um rótulo - a função `lv_label_set_style(label1, &style)` pode ser usada para configurar um novo estilo.

Se o objeto tem mais estilos (como um botão tem 5 estilos para cada estado) a função `lv_btn_set_style(obj, LV_BTN_STYLE_..., &rel_style` pode ser usada para configurar um novo estilo.

Os estilos e a propriedade de estilo usada por um tipo de objeto são descritas em sua documentação.

Se você **modificar um estilo na qual está sendo usado** por um ou mais objetos, ou objetos terão que ser notificados sobre o estilo que está mudado. Você tem duas opções para fazer que são:

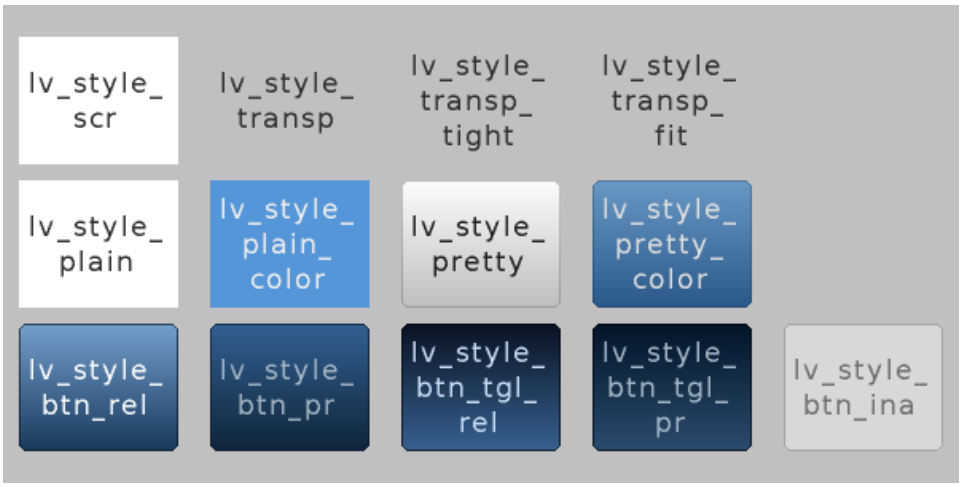
```
void lv_obj_refresh_style(lv_obj_t * obj);         /*Notifica um objeto sobre seus estilos que estão sendo modificado*/
void lv_obj_report_style_mod(void * style);         /*Notifica para todos os objetos se um estilo está sendo modificado. (NULL para no
tificar todos objetos)*/
```

Se o **estilo de um objeto é NULL, então seu estilo será herdado do estilo de seu pai**. Isso torna mais fácil criar um design consistente. Não se esqueça que um estilo descreve muita de suas propriedades ao mesmo tempo. Por exemplo, se você configurou um estilo de botão e criou um rótulo nele com estylo NULL, então o rótulo será renderizado de acordo com o estilo desse botão. Em outras palavas, o botão garante que seu filho se parecerá bem com ele.

Configurando a propriedade //glass irá evitar de herdar aquele estilo//. Você deve usá-lo se o estilo é transparente, então aquele filho usa as cores e outras de seu pai.

Estilo embutido

Existem muitos estilos embutidos na biblioteca:



Como você pode ver hpa um estilo para telas, para botões, planos e estilos bonitos e transparentes também. O `lv_style_transp`, `lv_style_transp_fit` e `lv_style_transp_tight` diferenciam somente em preenchimentos: para `lv_style_transp_tight` todos os preenchimentos são zeros, para `lv_style_transp_fit`, somente preenchimentos horizontais e verticais são zeros.

Os estilos embutidos são variáveis lv_style_t globais, então você pode usa-los como: lv_btn_set_style(obj, LV_BTN_STYLE_REL, &lv_style_btn_rel)

Você pode modificar os estilos embutidos ou você pode criar novos estilos. Quando criar novos estilos é recomendado primeiro copiar um estilo embutido para garantir que todos os campos são inicializados com valores apropriados. O lv_style_copy(&dest_style, &src_style) pode ser usado para copiar estilos.

Animações de estilo

Você pode animar estilos usando `lv_style_anim_create(&anim)`. Antes de chamar essa função você tem que inicializar uma variável `lv_style_anim_t`. A animação desaparecerá de um `style_1` para um `style_2`.

```
lv_style_anim_t a;    /*Será copiado para uma variável local*/
a.style_anim = & style_to_anim;     /*Ponteiro para um estilo a ser animado*/
a.style_start = & style_1;          /*Ponteiro para um estilo incial (somente o ponteiro salvo)*/
a.style_end = & style_2;            /*Ponteiro para o estilo alvo (somente o ponteiro salvo)*/
a.act_time = 0;                     /*Atribui negativo para fazer o atraso*/
a.time = 1000;                      /*Tempo da animaçãoo em milisegundos*/
a.playback = 0;                     /*1: Roda a animação de volta também*/
a.playback_pause = 0;               /*Espera antes de rodar a animação [ms]*/
a.repeat = 0;                       /*1: repetir a animação*/
a.repeat_pause = 0;                 /*Espera antes de repetir [ms]*/
a.end_cb = NULL;                    /*Chame essa animação quando estiver pronta*/
```

Exemplo de estilo

O exemplo de estilo abaixo mostra o uso do estilo descrito acima

```
/*Cria um estilo*/
static lv_style_t style1;
lv_style_copy(&style1, &lv_style_plain);    /*Copia um estilo embutido para iniciar o novo estilo*/
style1.body.main_color = LV_COLOR_WHITE;
style1.body.grad_color = LV_COLOR_BLUE;
style1.body.radius = 10;
style1.body.border.color = LV_COLOR_GRAY;
style1.body.border.width = 2;
style1.body.border.opa = LV_OPA_50;
style1.body.padding.hor = 5;            /*Preenchimento Horizontal usado para o indicador de barras abaixo*/
style1.body.padding.ver = 5;            /*Preenchimento usado para o indicador de barras abaixo*/
style1.text.color = LV_COLOR_RED;

/*Cria um objeto simples*/
lv_obj_t *obj1 = lv_obj_create(lv_scr_act(), NULL);
lv_obj_set_style(obj1, &style1);                        /*Aplica o estilo criado*/
lv_obj_set_pos(obj1, 20, 20);                           /*Atribui a posição*/

/*Cria um rótulo no objeto. O estilo do rótulo é NULL por padrão*/
lv_obj_t *label = lv_label_create(obj1, NULL);
lv_obj_align(label, NULL, LV_ALIGN_CENTER, 0, 0);       /*Alinha o rótulo no meio*/

/*Cria uma barra*/
lv_obj_t *bar1 = lv_bar_create(lv_scr_act(), NULL);
lv_bar_set_style(bar1, LV_BAR_STYLE_INDIC, &style1);    /*Modifica o estilo do indicador*/
lv_bar_set_value(bar1, 70);                             /*Atribui o valor da barra*/
```

Temas

Para criar estilos para seu GUI é um desafio porque você precisa de um profundo conhecimento da biblioteca e você precisa ter algumas habilidades de design. Em adição a isso, isto toma muito tempo para criar muitos estilos.

Para acelerar o projeto, alguns temas são introduzidos. Um tema é uma coleção de estilos na qual contém o estilo requerido para cada tipo de objeto. Por exemplo 5 estilos de botões para descrever 5 estados possíveis. Verifique os Temas existentes.

Para ser mais específico um tema é uma estrutura de variável na qual contém vários campos lv_style_t *. Para botões:

```
theme.btn.rel       /*Estilo de botão não pressionado*/
theme.btn.pr        /*Estilo de botão pressionado*/
theme.btn.tgl_rel   /*Estilo de botão alternado não pressionado*/
theme.btn.tgl_pr    /*Estilo de botão alternado pressionado*/
theme.btn.ina       /*Estilo de botão inativo*/
```

Um tema pode ser inicializado por: `lv_theme_xxx_init(hue, font)`. Onde xxx é o nome do tema, *hue* é uma valor de matiz do espaço de cor HSV (0..360) e a *font* é a fonte aplicada no tema (`NULL` para usar a fonte padrão `LV_FONT_DEFAULT`)

Quando um tema é inicializado seus estilos pode ser usados como esse:

```
/*Cria um deslizador padrão*/
lv_obj_t *slider = lv_slider_create(lv_scr_act(), NULL);
lv_slider_set_value(slider, 70);
lv_obj_set_pos(slider, 10, 10);

/*Inicializa o tema alien com uma matiz avermelhada*/
lv_theme_t *th = lv_theme_alien_init(10, NULL);

/*Cria um novo deslizador e aplica o estilo do tema*/
slider = lv_slider_create(lv_scr_act(), NULL);
lv_slider_set_value(slider, 70);
lv_obj_set_pos(slider, 10, 50);
lv_slider_set_style(slider, LV_SLIDER_STYLE_BG, th->slider.bg);
lv_slider_set_style(slider, LV_SLIDER_STYLE_INDIC, th->slider.indic);
lv_slider_set_style(slider, LV_SLIDER_STYLE_KNOB, th->slider.knob);
```

Você pode pedir para a biblioteca aplicar os estilos de um tema quando você cria novos objetos. Para fazer, use `lv_theme_set_current(th)`;

Dispositivos de entrada

Escrito para v5.1

Para interagir com o objeto criado o *Input devices* será requirido. Por exemplo touchpad, mouse, teclado ou até mesmo um codificador. Para aprender como adicionar uma entrada, leia o Guia de portabilidade.

Quando você registrar um dispositivo de entrada a biblioteca adiciona algumas informações extras para ela descrever o estado deste dispositivo em mais detalhes. Quando uma ação do usuário (ex.: pressionar botão) acontece uma função de ação (callback) é gatilhada e sempre há um dispositivo na qual gatilhou essa ação. Você pode obter esse dispositivo de entrada com

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

Isto pode ser inportante quando você precisa saber algumas informações especial sobre o dispositivo de entrada como o ponto pressionado frequentemente, ou arrastar um objeto ou não, etc.

Os dispositivos de entrada tem uma API muito simples:

```
/*Obter o último ponto na entrada do display*/
void lv_indev_get_point(lv_indev_t * indev, point_t * point);

/*Checa se existe arrasto no dispositivo de entrada ou não*/
bool lv_indev_is_dragging(lv_indev_t * indev);

/*Obtém o vetor do arrasto no dispositivo de entrada*/
void lv_indev_get_vect(lv_indev_t * indev, point_t * point);

/*Não faz nada até o próximo "deixar de pressionar"*/
void lv_indev_wait_release(lv_indev_t * indev);

/*Não faz nada até o próximo "deixar de pressionar"*/
void lv_indev_wait_release(lv_indev_t * indev);

/*Reseta um dispositivo de entrada ou todos (use NULL)*/
void lv_indev_reset(lv_indev_t * indev);

/*Reseta o estado de pressionamento longo de um dispositivo de entrada*/
void lv_indev_reset_lpr(lv_indev_t * indev);

/*Configura um curso para um ponteiro de um dispositivo de entrada*/
void lv_indev_set_cursor(lv_indev_t * indev, lv_obj_t * cur_obj);

/*Configura um grupo de destino para um dispositivo de entrada para um keypad*/
void lv_indev_set_group(lv_indev_t * indev, lv_group_t * group);
```

Navegação sem touchpad

Os objetos pode ser agrupado em ordem para facilmente **controlar** eles **sem touchpad ou mouse**. Ele permite a você usar

- Teclado ou keypad
- Botões de hardware
- Codificadores

para navegar entre objetos.

Primeiramente você tem que **criar um grupo de objeto** com `lv_group_t *group = lv_group_create()` e adicionar objetos para ele com `lv_group_add_obj(group, obj)`. Em um grupo sempre existe um objeto *focused*. Todos os botões pressionados serão "enviados" aos objetos focados.

Para navegar entre os objetos em um grupo (mundança no objeto focados) e interagir com eles, um tipo _LV_INDEV_TYPE_KEYPAD_ é requerido dispositivo de entrada. Nele a função *read* você pode pedir a biblioteca informar qual tecla é pressionada ou deixou de ser pressionada. Para aprender como adicionar um dispositivo de entrada, leia o Guia de portabilidade.

Assim você tem que **assinar ao grupo para um dispositivo de entrada** com

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

Existem alguns **controles de caracteres** especiais na qual podem ser usadas na função *read*:

- **LV_GROUP_KEY_NEXT** Foca no próximo objeto
- **LV_GROUP_KEY_PREV** Foca no objeto prévio
- **LV_GROUP_KEY_UP** Incrementa o valor, move acima ou clica no objeto focado (mover acima significa por exemplo selecionar um objeto acima em uma lista de elemento)
- **LV_GROUP_KEY_DOWN** Decrementa o valor ou mover para baixo no objeto focado (mover para baixo significa por exemplo selecionar um objeto abaixo na lista de elemento)
- **LV_GROUP_KEY_RIGHT** Incrementa o valor ou clique no objeto focado
- **LV_GROUP_KEY_LEFT** Decrementa o valor do objeto focado
- **LV_GROUP_KEY_ENTER** Clica no objeto focado ou em um elemento selecinado (ex: lista de elemento)
- **LV_GROUP_KEY_ESC** Fecha o objeto (ex: lista drop-down)

Em alguns casos (ex: quando uma janela pop-up aparece) é útil para congelar o foco sobre um objeto. Significa que _LV_GROUP_KEY_NEXT/PREV_ será ignorado. Você pode fazer isso com `lv_group_focus_freeze(group,true)`.

O **estilo do objeto em foco** é modificado por uma função. Por padrão, ele faz as cores do objeto alaranjada mas você pode especificar seu próprio estilo com uma função atualizadora em cada grupo com

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

O _style_mod_cb_ precisa de um parâmetro `lv_style_t *` na qual é uma cópia do estilo do objeto focado. Em um callback, você pode mixar algumas cores para o objeto em questão e modificar parâmetros mas **não é permitido atribuir "modificar o tamanho"** (como _letter_space_, *padding* etc.)

Cores

Escrito para v5.1

Os carregadores do módulo de cor, todas as funções de cores relacionadas como mudar profundidade de cor, criando cor apartir do código hexadecimal, converter profundidade de cor, mixando cores, etc.

Os tipos de variáveis seguintes são definidos pelo modulo da cor:

- **lv_color1_t** Guarda cor monocromática. Para compatibilidade também contém campos R,G,B mas eles são sempre os mesmos (1 Byte)
- **lv_color8_t** Uma estrutura para guardar componentes R (3 bit), G (3 bit), B (2 bit) para cores de 8 bits (1 Byte)
- **lv_color16_t** Uma estrutura para guardar componentes R (5 bit), G (6 bit), B (5 bit) para cores de 16 bits (2 Bytes)
- **lv_color24_t** Uma estrutura para guardar componentes R (8 bit), G (8 bit), B (8 bit) para cores de 24 bits (4 Bytes)
- **lv_color_t** Igual para color 1/8/16/24_t de acordo para as configurações de profundidade
- **lv_color_int_t** uint8_t, uint16_t ou uint32_t de acordo para a configuraçõa de profundidade de cor. Usada para construir vetores de cor através de números planos.
- **lv_opa_t** Uma tipo simples uint8_t para descrever a opacidade.

Os tipos _lv_color_t_, _lv_color1_t__lv_color8_t_, _lv_color16_t_ e _lv_color24_t_ tem 4 campos:

- **red** Canal vermelho
- **green** Canal verde
- **blue** Canal azul
- **full** vermelho + verde + azul como um número

Você pode configurar **comprimento de cor atual** em _lv_conf.h_ atualizando e definindo o _LV_COLOR_DEPTH_ para 1 (monocromático), 8, 16 ou 24.

Você pode **converter uma cor de uma profundidade de cor atual** para um outro. As funções conversoras retorna com um número, então você pode usar o campo *full*:

```
lv_color_t c;
c.red   = 0x38;
c.green = 0x70;
c.blue  = 0xCC;

lv_color1_t c1;
c1.full = lv_color_to1(c);        /*Retorna 1 para cores de luzes, 0 para cores escuras*/

lv_color8_t c8;
c8.full = lv_color_to8(c);        /*Dá um número de 8 bit com a cor convertida*/

lv_color16_t c16;
c16.full = lv_color_to16(c);     /*Dá um número de 16 bit com a cor convertida*/

lv_color24_t c24;
c24.full = lv_color_to24(c);    /*Dá um número de 32 bit com a cor convertida*/
```

Você pode **criar uma cor** com a profundidade de cor atual usando a macro **LV_COLOR_MAKE**. Ele exige 3 argumentos (vermelho, verde, azul) como número de 8 bit. Por exemplo, para criar uma cor vermelho claro: `my_color = COLOR_MAKE(0xFF,0x80,0x80)`. Cores podem ser criadas de **código HEX** também: `my_color = LV_COLOR_HEX(0xFF8080)` ou `my_color = LV_COLOR_HEX3(0xF88)`.

Mixando duas cores é possível com `mixed_color = lv_color_mix(color1, color2, ratio)`. A razão pode ser 0..255. 0 (zero) resulta completamente color2, 255 resulta completamente color1.

Para descrever a **opacidade** o tipo _lv_opa_t_ é criado como embrulho para _uint8_t_. Algumas definições são introduzidos:

- **LV_OPA_TRANSP** Valor: 0, significa a opacidade faz a cor completamente transparente
- **LV_OPA_10** Valor: 25, significa que a cor cobre um pouco
- **LV_OPA_20 ... OPA_80** segue a lógica

- **LV_OPA_90** Valor: 229, significa que a cor está perto de cobrir completamente
- **LV_OPA_COVER** Valor: 255, significa que a cor cobre completamente

Você pode também usar a _LV_OPA_*_ definido em _lv_color_mix() como razão._

O módulo de cor define as **cores mais básicas**:

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

como também `LV_COLOR_WHITE`.

Fontes

Escrito para v5.1

As fontes LittlevGL são bitmaps e outros descritores para guardar as imagens das letras (glyph) e algumas informações adicionais. Uma fonte é armazenada em uma variável **lv_font_t** e pode ser configurado no campo de estilo *text.font*.

As fontes tem uma propriedade **bpp (Bit-Per-Pixel)**. Ele mostra como o bit é usado para descrever um pixel na fonte. O valor armazenado para um pixel determina a opacidade do pixel. O jeito que a imagem das letras (especialmente sobre as bordas) podem ser suavizada. O valor bpp são 1, 2, 4 e 8 (maior valor significa melhor qualidade). O bpp também afeta o tamanho da memória requirida para armazenar a fonte. Ex.: bpp = 4 faz a memória da fonte 4 vezes maior do que bpp = 1.

Fontes embutidas

Existe muitas fontes embutidas na qual podem ser ativadas em lv_conf.h pelas definições _USE_LV_FONT_..._. Exinte fontes embutidas em **estilos diferentes**:

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

Você pode ativar as fontes com valores 1, 2, 4 ou 8 para configurar seus bpp (ex.: `#define USE_LV_FONT_DEJAVU_20 4` no `lv_conf.h`).

As fontes embutidas existem com **multiplicar caracteres** em cada tamanho:

- ASCII (Unicode 32..126)
- Suplemento Latim (Unicode 160..255)
- Cirilico (Unicode 1024..1279)

As fontes embutidas usa a fonte *Dejavu*.

As fontes embutidas são **variáveis globais** com nomes como:

- `lv_font_dejavu_20` (fonte ASCII de 20 px)
- `lv_font_dejavu_20_latin_sup` (Fontes suplementar em Latin de 20 px)
- `lv_font_dejavu_20_cyrillic` (Fonte cirílico de 20 px)

Suporte unicode

O LittlevGL suporta letras Unicode apartir da codificação de caracteres **UTF-8**. Você precisa configurar seu editor para salvar seu código/texto como UTF-8 (usualmente este é o padrão) e ativado _LV_TXT_UTF8_ no lv_conf.h. Sem a ativação do _LV_TXT_UTF8_ somente fontes ASCII e símbolos pode ser usado (veja os símbolos abaixo)

Depois disso os textos serão decodificados para determinar os valores Unicode. Para mostrar as letras sua fonte precisa conter a imagem (glyph) dos caracteres.

Você pode assinar mais fontes para criar um **conjunto de caracteres maior**. Para fazer isso escolha a fonte base (tipicamente fonte ASCII) e adiciona a extensão a ele `lv_font_add(child, parent)`. Somente fonte com mesmo tamanho serão assinadas.

As fontes embutidas já estão adicionadas ao mesmo tamanho da fonte ASCII. Por exemplo, se estão ativados os _USE_LV_FONT_DEJAVU_20_ e _USE_LV_FONT_DEJAVU_20_LATIN_SUP_ dentro do `lv_conf.h` então o texto *"abcÁÖÜ"* pode ser renderizado quando usar _lv_font_dejavu_20_.

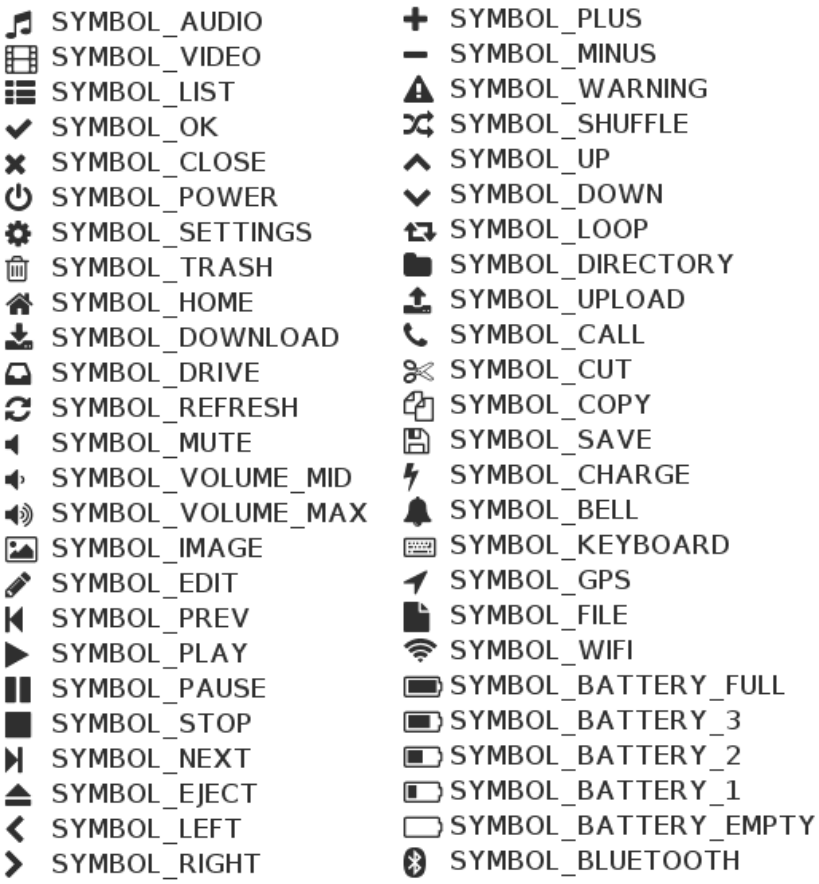
Fontes de símbolos

O fonte de símbolos são fontes especiais na qual contém símbolos ao invés de letras. Há **fontes símbolos nas fontes embutidas** também e elas são assinadas para fontes ASCII do mesmo tamanho. Em um texto, um símbolo pode ser referenciado como _SYMBOL_LEFT_, _SYMBOL_RIGHT_ etc; VocÇe pode mixar estes nomes de símbolos com strings:

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

Os símbolos podem ser usado sem suporte UTF-8 também. (_LV_TXT_UTF8 0_)

As listas acima mostra os símbolos existentes:



Adicionar nova fonte

Se voccê quer **adicionar nova fonte para a biblioteca** você pode usar a Ferramenta de Conversão de Fonte. Ele pode criar um array em C de um arquivo TTF na qual pode ser copiado para o seu projeto. Você pode especificar o tamanho, o alcance dos caracteres e o bpp. Opcionalmente você pode enumerar os caracteres para incluir eles na fonte final. Para usar a fonte gerada, declare isso com _LV_FONT_DECLARE(my_font_name)_. Depois disto, a fonte pode ser usada como fonte embutida.

Fonte de exemplo

```
/*Cria um novo estilo para o rótulo*/
static lv_style_t style;
lv_style_copy(&style, &lv_style_plain);
style.text.color = LV_COLOR_BLUE;
style.text.font = &lv_font_dejavu_40;   /*Fontes e símbolos unicode já assinadas para a biblioteca*/

lv_obj_t *label;

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

/*Misturando textos e símbolos*/
label = lv_label_create(lv_scr_act(), NULL);
lv_obj_set_pos(label, 20, 100);
lv_label_set_style(label, &style);
lv_label_set_text(label, "Right "SYMBOL_RIGHT);
```

Desenhos

Escrito para v5.1

No *LittlevGL* você pode pensar em objetos gráficos e não precisa se preocupar de como os desenhos acontecem. Você pode configurar o tamanho, posição ou qualquer atributo do objeto e aa biblioteca irá atualizar as áreas antigas (inválidas) e redesenhará as novas. Contudo, você deve saber o básico dos métodos de desenho para saber em qual deva escolher.

Desenho buferizado e sem buffer

Desenho sem buffer

Os desenhos sem buffer coloca os pixels diretamente no display (frame buffer). Contudo durante o processo de desenho, alguma cintilação pode ser visível porque primeiramente o background tem que ser desenhado e depois os objetos sobre ele. Por essa razão, este tipo não está disponível quando deslizar, arrastar e animações são usadas. Por outro lado, ele tem a menor memória porque nenhum buffer extra para gráfico é requerido

Para usar desenho não bufferizado configure _LV_VDB_SIZE_ para 0 dentro do _lv_conf.h_ e registre o _disp_map_ e _disp_fill_functions. Aqui você pode aprender mais sobre Portabilidade.

Desenho bufferizado

O desenho bufferizado é similar a bufferização dupla quando duas telas com buffer dimensionado são usados (um para renderização e outro para mostrar o último quadro disponível). Contudo, o algoritmo do desenho bufferizado do LittlevGL usa somente um frame buffer e um pequeno buffer chamado Virtual Display Buffer (VDB). Para o tamanho do VDB ~1/10 do tamanho da tela é tipicamente o suficiente. Para uma tela de 320 x 240 com cores de 16-bit, é necessário somente 15 KB de RAM extra.

Com o desenho bufferizado não há cintilação porque a imagem é criada primeiramente na memória (VDB), contudo deslizar, arrastas e animações podem ser usadas. Em adição, ela também ativa o uso de outros efeitos gráficos como anti-aliasing, transparencia (opacidade) e sombras.

Para usar desenho bufferizado configure _LV_VDB_SIZE_ para > LV_HOR_RES dentro do _lv_conf.h_ e registre uma função _disp_flush_.

No modo bufferizado, você pode usar **VDB duplo** para paralelamente executar renderização dentro de um VDB e copiar pixels para seu frame buffer de um para outro. A cópia deve usar DMA ou outra aceleração de hardware para funcionar em background para deixar a CPU fazer outras tarefas. Dentro do _lv_conf.h_ o _LV_VDB_DOUBLE 1_ ativa esta característica.

Desenho bufferizado vs desenho não bufferizado

Mantenha em mente que não é garantido que o desenho não bufferizado é mais rápido. Durante o processo de renderização, um pixel é sobrescrevido múltiplas vezes (ex.: backgroundm botão, texto estão acima um dos outros). Deste jeito em um modo não bufferizado a biblioteca precisa acessar a memória externa ou o controlador do display múltiplas vezes na qual é mais lento do que escrever/ler a memória interna da RAM.

As tabelas seguintes resume as diferenças entre os dois métodos de desenho:

	Des. não buffer.	**Des. bufferizado**
Uso da memória	Sem extra	>~1/10 da tela
Qualidade	Cintilização	Sem falha
Antialiasing	Não suportado	Suportado
Transparência	Não suportado	Suportado
Sombras	Não suportado	Suportado

Anti-aliasing

Dentro do lv_conf.h você pode ativar o anti-aliasing com _LV_ANTIALIAS 1_. O anti-aliasing é suportado somente no mode bufferizado (LV_VDB_SIZE > LV_HOR_RES).

Os algoritmo anti-aliasing coloca alguns pixels transluzente (pixels com opacidade) para produzir linhas e curvas (incluindo bordas com raio) e até mesmo suavizamento. Porque ele coloca somente pixels anti-aliasing que requer somente um pouco de poder computacional extra (~1.1x do tempo extra comparado ao não configuração anti-aliasing)

Como descrito em Seção de fonte as fontes podem ser anti-aliased por usar uma fonte diferente com um bpp (Bit-Per-Pixel) maior. Deste jeito os pixels de uma fonte pode não somente ser 0 ou 1 mas ser transluzente. Os bpp-s suportados são 1, 2, 4 e 8. Mantenha na mente uma fonte com um maior bpp requer mais ROM.

Animações

Escrito por v5.1

Você pode automaticamente mudar o valor (animar) de uma variável entre um início e um valor fim usando **função animar** com o protótipo `void func(void* var,int32_t value). A animação acontecerá por uma chamada periódica da função animadora com o parâmetro correspondente.

Para **criar uma animação** você tem que inicializar uma variável _lv_anim_t_ (existe um template em lv_anim.h):

```
lv_anim_t a;
a.var = button1;                          /*Variável para animar*/
a.start = 100;                             /*Valor inicial*/
a.end = 300;                              /*Valor final*/
a.fp = (lv_anim_fp_t)lv_obj_set_height;                              /*Função para ser usada para animar*/
a.path = lv_anim_path_linear;                                         /*Caminho da animação*/
a.end_cb = NULL;                          /*Callback quando a animação estiver pronta*/
a.act_time = 0;                            /*Configura o tempo < 0 para um atraso [ms]*/
a.time = 200;                             /*Comprimento da animação [ms]*/
a.playback = 0;                            /*1: anima em direção reversa também quando o normal estiver pronto*/
a.playback_pause = 0;                           /*Espera antes do playback [ms]*/
a.repeat = 0;                             /*1: Repete a animação (com ou sem playback)*/
a.repeat_pause = 0;                        /*Espera antes de repetir [ms]*/

lv_anim_create(&a);                       /*Inicia a animação*/
```

A `anim_create(&a)` irá registrar a animação e imediatamente **aplicar o valor *start*** independentemente para configurar o atraso.

Você pode determinar o **atalho da animação**. Na maioria simples do caso ele é linear na qual significa os valores corrente *start* e *end* é mudado linearmente. O atalho é uma função na qual calcula o próximo valor para configurar baseado no estado corrente da animação. Correntemente há dois atalhos embutidos:

- **lv_anim_path_linear** animação linear
- **lv_anim_path_step** muda em um passo no fim

Por padrão, você pode configurar o tempo da animação. Mas em alguns casos, a **velocidade de animação** é mais prática. A função `lv_anim_speed_to_time(speed, start, end)` calcula o tempo requerido em milisegundos para alcançar o fim do valor apartir de um valor inicial com a dada velocidade. A velocidade é interpretado em dimensão *unit/sec*. Por exemplo `lv_anim_speed_to_time(20,0,100)` nos dará 5000 milisegundos.

Você pode aplicar **animações diferentes múltiplas** na mesma variável no mesmo tempo. (Por exemplo, animar as coordenadas x e y com _lv_obj_set_x_ e _lv_obj_set_y_). Mas somente uma animação pode existir com um uma dada variável e um par de funções. Contudo a função _lv_anim_create()_ apagará a função existente de animação.

Você pode **apagar uma animação** por `lv_anim_del(var, func)` com a variável animada providenciada e sua função animadora.

Guia de estilo de código

Revisão 2

Formato de arquivo

Use lv_misc/lv_templ.c e lv_misc/lv_templ.h

Convenção de nomes

- Palavras são separados por '_'
- Em variável e nomes de funções os nomes estão em minúsculas (ex.: *height_tmp*)
- Em enumerações e definições, usar somente maiúsculas (ex.: MAX_LINE_NUM*)
- Nomes globais (API):
  - começa com *lv*
  - seguido por nome de módulo: *btn*, *label*, *style* etc.
  - seguido pela ação (para funções): *set*, *get*, *refr* etc.
  - fechado com o sujeito: *name*, *size*, *state* etc.
- Typedefs
  - preferir `typedef struct` e `typedef enum` ao invés de `struct name` e `enum name`
  - sempre finalize `typedef struct` e `typedef enum` que finalize com `_t`
- Abreviações:
  - Use abreviações em nomes públicos somente se eles se tornarem maiores que 32 caracteres
  - Use somente muita abreviações diretas como (e.g. pos: posição) ou abreviações bem estabelecidas (ex.: pr: pressionar).

Guia de codigo

- Funções:
  - Tente escrever funções menores que 50 linhas
  - Sempre menor que 100 linhas (exceto algo mais aprimorado)
- Variáveis:
  - Uma linha, uma declaração (RUIM: char x, y;)
  - Use `<stdint.h>` (*uint8_t*, *int32_t* etc)
  - Declare variáveis quando necessário (nem todas no início da função)
  - Use o menor escopo requerido
  - Variáveis dentro de um arquivo (fora das funções) serão sempre *static*
  - Não use variáveis globais (use funções para configurar/obter variáveis estáticas)

Comentários

Antes de toda função ter um comentário como este:

```
/**
 * Rerotna com a tela de um objeto
 * @param obj aponta para um objeto
 * @return ponteiro para uma tela
 */
lv_obj_t * lv_obj_get_scr(lv_obj_t * obj);
```

Use sempre o formato `/* Alguma coisa */` e NÃO `//Alguma coisa`

Escreva o código escrito para evitar comentários descritivos como: `x++; /* Adiciona 1 a x */`. O código deve mostrar claramente o que você está fazendo.

Você deve escrever **por que** você fez isso: `x++; /*Porque ao fechar '\0' de um string*/`

"Código resumidos" curtos de umas poucas linhas são aceitáveis. Ex.: `/*Calcular as novas coordenadas*/`

Nos comentários use ` ` quando referenciar para uma variável. Ex.: `/*Atualizar o valor de `x_act`*/`

Formatando

Aqui é exemplo de como aplicar colchetes e usar espaços em branco:

```
/**
 * Configura um novo texto para um rótulo. Memória será alocada para guardar o texto para um rótulo.
 * @param ponteiro do rótulo para um rótulo do objeto
 * @param texto terminado com string de caractere '\0'. NULL para atualizar o texto atual
 */
void lv_label_set_text(lv_obj_t * label, const char * text)
{   /* Colchetes principais da função em nova linha*/

    if(label == NULL) return; /*Nenhum colchete, somente se o comando em linha com a condição if*/

    lv_obj_inv(label);

    lv_label_ext_t * ext = lv_obj_get_ext(label);

    /*Comente antes uma seção*/
    if(text == ext->txt || text == NULL) {  /*Colchete de uma condiçção começa em linha*/
        lv_label_refr_text(label);
        return;
    }

    ...
}
```

Use identação de 4 espaços ao invés de tab.

Você pode usar **astyle** para formatar o código. As configurações requeridas estão: `docs/astyle_c` e `docs/astyle_h` Para formatar os arquivos dos códigos: `$ find . -type f -name "*.c" | xargs astyle --options=docs/astyle_c`

Para formatar os arquivos de cabeçalho: `$ find . -type f -name "*.h" | xargs astyle --options=docs/astyle_h`

Adicione `-n` para o fim para pular a criação de arquivo de backup OU use `$ find . -type f -name "*.bak" -delete` (para backups de arquivo fonte) and `find . -type f -name "*.orig" -delete` (para backup de arquivos de cabeçalho)

Tipos de objetos

Escrito para V5.1

As seguintes páginas contém documentações detalhadas para cada objeto dentro do Littlev Graphics Library.

- Objeto base (lv_obj)
- Rótulo (lv_label)
- Imagem (lv_img)
- Linha (lv_line)
- Arco (lv_arc)
- Container (lv_cont)
- Página (lv_page)
- Janela (lv_window)
- Vista de tabulação (lv_tabview)
- Barra (lv_bar)
- Medidor de linha (lv_lmeter)
- Indicador de medida (lv_gauge)
- Gráfico (lv_chart)
- LED (lv_led)
- Pré-carregador (lv_preload)
- Caixa de mensagem (lv_mbox)
- Área de texto (lv_ta)
- Calendário (lv_calendar)
- Botão (lv_btn)
- Imagem de botão (lv_imgbtn)
- Matriz de botão (lv_btnm)
- Teclado (lv_kb)
- Lista (lv_list)
- Lista drop down (lv_ddlist)
- Rolo (lv_roller)
- Caixa de checagem (lv_cb)
- Interruptor (lv_sw)
- Controle deslizandte (lv_slider)

Arco (lv_arc)

Escrito para v5.2

Visão geral

O objeto Arco **desenha um arco** em torno **dos ângulos iniciais e finais** e com uma dada **espessura**.

Para configurar os ângulos, use a função `lv_arc_set_angles(arc, start_angle, end_angle)`. O grau zero está no fundo do objeto e os graus aumentam em um sentido anti-horário. Os ângulos devem estar entre [0;360].

Para **configurar o estilo** de um objeto Arco, use `lv_arc_set_style(arc, LV_ARC_STYLE_MAIN, &style)`

Uso do estilo

- **line.rounded** faz os as linhas arredondadas (opacidade não funcinará apropriadamente se configurado para 1)
- **line.width** a espessura do arco
- **line.color** a cor do arco.

Notas

- O **comprimento e altura** do Arco devem ser os **mesmos**
- Atualmente o objeto Arco **não suporta anti-aliasing**.

Exemplo

```
/*Cria o estilo para os Arcos*/
lv_style_t style;
lv_style_copy(&style, &lv_style_plain);
style.line.color = LV_COLOR_BLUE;           /*Cor do Arco*/
style.line.width = 8;                       /*Comprimento do Arco*/

/*Cria um Arco*/
lv_obj_t * arc = lv_arc_create(lv_scr_act(), NULL);
lv_arc_set_style(arc, LV_ARC_STYLE_MAIN, &style);          /*Usa o novo estilo*/
lv_arc_set_angles(arc, 90, 60);
lv_obj_set_size(arc, 150, 150);
lv_obj_align(arc, NULL, LV_ALIGN_CENTER, 0, 0);

/*Copia o Arco anterior e configura ângulos diferentes e tamanho*/
arc = lv_arc_create(lv_scr_act(), arc);
lv_arc_set_angles(arc, 90, 20);
lv_obj_set_size(arc, 125, 125);
lv_obj_align(arc, NULL, LV_ALIGN_CENTER, 0, 0);

/*Copia o Arco anterior e configura diferente ângulos e tamanho*/
arc = lv_arc_create(lv_scr_act(), arc);
lv_arc_set_angles(arc, 90, 310);
lv_obj_set_size(arc, 100, 100);
lv_obj_align(arc, NULL, LV_ALIGN_CENTER, 0, 0);
```

Barra (lv_bar)

Escrito para v5.1

Visão global

Os objetos Barra tem duas partes principais: um **background** na qual é próprio objeto e um **indicador** na qual a forma é similar ao plano de fundo (background) mas seu comprimento/altura pode ser ajustado.

A orientação da barra pode sser **vertical ou horizontal** de acordo com a razão comprimento/altura. Logicamente nas barras horizontais o comprimento do indicador, sobre as barras verticais a altura d indicador pode ser mudado.

Um **novo valor** pode ser ajustado por: `lv_bar_set_value(bar, new_value)`. O valor é interpretado em **alcance** (valores mínimos e máximos) na qual podem ser modificados com `lv_bar_set_range(bar, min, max)`. O alcance padrão é: 1..100.

The setting of a new value can happen with an **animation** from the current value to the desired. In this case use `lv_bar_set_value_anim(bar, new_value, anim_time)`.

Uso do estilo

- **background** é um Base-object contudo ele usa seus elementos de estilos. Seu estilo padrão é: `LV_STYLE_PRETTY`.
- **indicador** é similar ao background. Seu estilo pode ser configurado por: `lv_bar_set_style_indic(bar,&style_indic)`. Ele usa o estilo de elemento *hpad* e *vpad* para manter o espaço do background. Seu estilo padrão é: `LV_STYLE_PRETTY_COLOR`.

Notas

- O indicador não é um objeto real; ele é somente desenhado pela barra.

Exemplo

```c
/*Cria uma barra padrão*/
lv_obj_t * bar1 = lv_bar_create(lv_scr_act(), NULL);
lv_obj_set_size(bar1, 200, 30);
lv_obj_align(bar1, NULL, LV_ALIGN_IN_TOP_RIGHT, -20, 30);
lv_bar_set_value(bar1, 70);

/*Cria um rótulo à direita da barra*/
lv_obj_t * bar1_label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(bar1_label, "Default");
lv_obj_align(bar1_label, bar1, LV_ALIGN_OUT_LEFT_MID, -10, 0);

/*Cria uma barra e um estilo indicador*/
static lv_style_t style_bar;
static lv_style_t style_indic;

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

lv_style_copy(&style_indic, &lv_style_pretty);
style_indic.body.grad_color =  LV_COLOR_GREEN;
style_indic.body.main_color=  LV_COLOR_LIME;
style_indic.body.radius = LV_RADIUS_CIRCLE;
style_indic.body.shadow.width = 10;
style_indic.body.shadow.color = LV_COLOR_LIME;
style_indic.body.padding.hor = 3;           /*Faz o indicador ficar um pouco menor*/
style_indic.body.padding.ver = 3;

/*Cria uma segunda barra*/
lv_obj_t * bar2 = lv_bar_create(lv_scr_act(), bar1);
lv_bar_set_style(bar2, LV_BAR_STYLE_BG, &style_bar);
lv_bar_set_style(bar2, LV_BAR_STYLE_INDIC, &style_indic);
lv_obj_align(bar2, bar1, LV_ALIGN_OUT_BOTTOM_MID, 0, 30); /*Alinha abaixo da 'barra1'*/

/*Cria uma segunda barra*/
lv_obj_t * bar2_label = lv_label_create(lv_scr_act(), bar1_label);
lv_label_set_text(bar2_label, "Modified");
lv_obj_align(bar2_label, bar2, LV_ALIGN_OUT_LEFT_MID, -10, 0);
```

Objeto-base (lv_obj)

Escrito para v5.1

Visão geral

O objeto base contém os **atributos mais básicos** dos objetos:

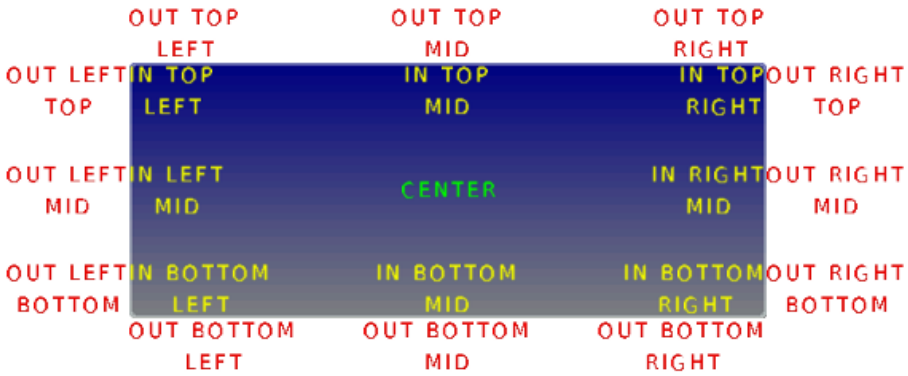
- Coordenadas
- Objeto pai
- Filhos
- Estilo
- Atributos como ativar Clique, ativar Arrasto etc.

Você pode atribuit a **coordenadas x e y** relativa ao pai com `lv_obj_set_x(obj, new_x)` e `lv_obj_set_y(obj, new_y)` ou em uma função com `lv_obj_set_pos(obj, new_x, new_y)`.

O **tamanho do objteto** pode ser modificado com `lv_obj_set_width(obj, new_width)` e `lv_obj_set_height(obj, new_height)` ou em uma função com `lv_obj_set_size(obj, new_width, new_height)`.

Você pode **alinhar** o objeto para um outro `lv_obj_align(obj1, obj2, LV_ALIGN_TYPE, x_shift, y_shift)`. O último dois argumentos significa um desllocamento x e y depois do alinhamento. O segundo argumento é outro objeto na qual alinha o primeiro (`NULL` significa: alinhamento ao pai).

O terceiro argumento é o tipo de alinhamento:



Os tipos de alinhamento construídos como `LV_ALIGN_OUT_TOP_MID`. Por exemplo alinha um texto abaixo de uma imagem: `lv_obj_align(text, image, LV_ALIGN_OUT_BOTTOM_MID, 0, 10)`. Ou para alinhar um texto no meio de seus pais: `lv_obj_align(text, NULL, LV_ALIGN_CENTER, 0, 0)`.

Você pode atribuir um **novo pai** para um objeto como `lv_obj_set_parent(obj, new_parent)`.

Para obter o filho de um objeto use `lv_obj_get_child(obj, child_prev)` (do últimmo ao primeiro) ou `lv_obj_get_child_back(obj, child_prev)` (do primiro ao último). Para obter o primeiro filho passe *NULL* como o segundo parâmetro e então os filhos prévios (retornam valores). A função irá retornar *NULL* quando não mais existir nenhum filho.

Quando você tiver criado uma **tela** como `lv_obj_create(NULL, NULL)` você pode **carregar** ele com `lv_scr_load(screen1)`. A `lv_scr_act()` função te dá um ponteiro para a **tela atual**

Existem **duas camadas** geradas automaticamente:

- camada de topo
- camada do sistema

Eles são independentes das telas assim objetos criados tal que essas camadas será mostradas em todas as telas. A *camada do topo* está acima de todos objetos da tela e a *camada do sistemma* está na camada topo também. Você pode adicionar qualquer *camada de topo* de janela pop-up livremente. Mas a *camada de sistema* estará restrito as coisas do nível do sistema (ex.: cursor do mouse será movido aqui). As funções `lv_layer_top()` e `lv_layer_sys()` dá um ponteiro para a camada de topo ou sistema.

Você pode configurar **novo estilo** para um objeto com a função `lv_obj_set_style(obj, &new_style)`. Se `NULL` é atribuido como estilo então o objeto irá inserir seu estilo do pai. Se você **modificar um estilo** você tem que **notificar os objetos** na qual estão usando o sistema modificado.

Você pode usar também `lv_obj_refresh_style(obj)` ou para notificar todos objetos com um dado estilo `lv_obj_report_style_mod(&style)`. Atribui o parâmetro _lv_obj_report_style_mod_' para `NULL` para notificar todos os objetos.

Há alguns atributos na qual podem ser ativados/desativados por `lv_obj_set_...(obj, true/false)`:

- **hidden** Esconde o objeto. Ele não irá ser desenhado e não ocupará espaço. Seus filhos serão escondidos também.
- **click** Ativado para o objeto clique via um dispositivo de entrada (ex.: touch pad). Se desativado então o objeto atrás dele irá ser checado durante o clique no carregador do dispositivo de entrada (útil com objetos não clicáveis como Rótulos)
- **top** Se ativado então quando esse objeto ou qualquer de seus filhos forem clicados então esse objeto vem para frente.
- **drag** Etiva o arrasto (movendo por um dispositivo de entrada)
- **drag_throw** Ativa "lançamento" com arrasto como o objeto tivesse momento
- **drag_parent** Se ativado então o pai do objeto será movido durante o arrasto.

Há algumas ações específicas na qual acontecem automaticamente na biblioteca. Para prevenir um ou mais deste tipos de ações você pode **proteger o objeto**. As seguintes proteções são:

- **LV_PROTECT_NONE** Sem proteção
- **LV_PROTECT_POS** Evita posicionamento automático (Ex.: Layout em lv_cont)
- **LV_PROTECT_FOLLOW** Evita que o objeto seja seguido em ordem automática (ex.: Layout em lv_cont)
- **LV_PROTECT_PARENT** Evita mudança automática do pai
- **LV_PROTECT_CHILD_CHG** Desativa a mudança do sinal do filho. Usado pela biblioteca.

O `lv_obj_set/clr_protect(obj, LV_PROTECT_...)` atribui/limpa a proteção. Você pode usar valores *'OR'ed* de tipos de proteção também.

Há **animações embutidas** para os objetos. As seguintes animações são:

- **LV_ANIM_FLOAT_TOP** Flutua do/para o topo
- **LV_ANIM_FLOAT_LEFT** Flutua da/para a esquerda
- **LV_ANIM_FLOAT_BOTTOM** Flutua de/para o fundo
- **LV_ANIM_FLOAT_RIGHT** Flutua de/para a direita
- **LV_ANIM_GROW_H** Cresce/encolhe horizontalmente
- **LV_ANIM_GROW_V** Cresce/encolhe verticalmente

O lv_obj_animate(obj, anim_type, time, delay, callback)` aplica uma animação sobre *obj*. Para determinar a direção da animação _OR_ _ANIM_IN_ ou _ANIM_OUT_ com o tipo de animação. O padrão é _ANIM_IN_ se não especificado. Você pode aprender mais sobre a animations.

Uso do estilo

Todas as propriedades *style.body* são usadas. Os padrões para telas _lv_style_plain_ e _lv_style_plain_color_ para objetos normais

Examplo

```
/*Cria um simples objeto de base*/
lv_obj_t * obj1;
obj1 = lv_obj_create(lv_scr_act(), NULL);
lv_obj_set_size(obj1, 150, 40);
lv_obj_set_style(obj1, &lv_style_plain_color);
lv_obj_align(obj1, NULL, LV_ALIGN_IN_TOP_MID, 0, 40);

/*Copia o objeto anterior e ativa o arrasto*/
lv_obj_t * obj2;
obj2 = lv_obj_create(lv_scr_act(), obj1);
lv_obj_set_style(obj2, &lv_style_pretty_color);
lv_obj_set_drag(obj2, true);
lv_obj_align(obj2, NULL, LV_ALIGN_CENTER, 0, 0);

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

/*Copia o objeto anterior (arrasta se já estiver ativado)*/
lv_obj_t * obj3;
obj3 = lv_obj_create(lv_scr_act(), obj2);
lv_obj_set_style(obj3, &style_shadow);
lv_obj_align(obj3, NULL, LV_ALIGN_IN_BOTTOM_MID, 0, -40);
```

Botão (lv_btn)

Escrito para v5.1, revisão 2

Visão geral

Botões podem reagir quando o usuário **pressiona**, **solta** ou **pressiona longamente**, via funções de callback (`lv_action_t` function pointers). Você pode configurar os funções callback com: `lv_btn_set_action(btn, ACTION_TYPE, callback_func)`. Os possíveis tipos de ação são:

- LV_BTN_ACTION_CLICK: o botão é solto depois de pressionado (clicado) ou, quando usar o keypad, depois que a tecla `LV_GROUP_KEY_ENTER` é solta
- LV_BTN_ACTION_PR: o botão é pressionado
- LV_BTN_ACTION_LONG_PR: o botão é pressionado longamente
- LV_BTN_ACTION_LONG_PR_REPEAT: o botão é pressionado e essa ação é gatilhada periodicamente

Botões podem ser um dos **cinco possíveis estados**:

- LV_BTN_STATE_REL Estado solto
- LV_BTN_STATE_PR Estado pressionado
- LV_BTN_STATE_TGL_REL Estado comutado (Estado Ligado)
- LV_BTN_STATE_TGL_PR Estado comutado pressionado (Estado Ligado pressionado)
- LV_BTN_STATE_INA Estado inativo

Os botões podem ser configurados como **botões comutados** com `lv_btn_set_toggle(btn, true)`. Nesse caso ao soltar, o botão vai pra o estado comutado solto.

Você pode configurar os estados dos botões manualmente através do: `lv_btn_set_state(btn, LV_BTN_STATE_TGL_REL)`.

Um botão pode ir para **Estado inativo** somente manualmente (através do _lv_btn_set_state()_). Em um Estado inativo, nehuma das ações será chamada.

Similarmente para Containers botões também possuem **layout** e **auto ajuste**:

- Configuração de layout `lv_btn_set_layout(btn, LV_LAYOUT_...)`. O padrão é LV_LAYOUT_CENTER. Assim, se você adicionar um rótulo ele será automaticamente alinhado ao meio.
- `lv_btn_set_fit(btn, hor_en, ver_en)` ativa para configurar o comprimento do botão e/ou altura automaticamente de acordo com os filhos.

Uso de estilos

Um botão pode ter 5 estilos independentes para os 5 estados. Você pode configurá-los via: `lv_btn_set_style(btn, LV_BTN_STYLE_..., &style)`. Os estilos usam a propriedade *style.body*.

- **LV_BTN_STYLE_REL** estilo do estado solto. Padrão: _lv_style_btn_rel_
- **LV_BTN_STYLE_PR** estilo do estado pressiondo. Padrão: _lv_style_btn_pr_
- **LV_BTN_STYLE_TGL_REL** estilo do estado comutado. Padrão: _lv_style_btn_tgl_rel_
- **LV_BTN_STYLE_TGL_PR** estilo do estado comutado pressionado. Padrão: _lv_style_btn_tgl_pr_
- **LV_BTN_STYLE_INA** estilo do estado inativo. Padrão: _lv_style_btn_ina_

Notas

- Se um botão é arrastado seu clique e pressionamento longo as ações não serão chamadas
- Se um botão foi pressionado longamente e sua ação longa ao pressionar foi configurada então sua ação clique não será chamada

Exemplo


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

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

    /* O botão é solto.
     * Faça algo aqui */

    return LV_RES_OK; /*Retorna OK se o botão não está apagado*/
}

.
.
.

/*Cria um rótulo para o título*/
lv_obj_t * label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(label, "Default buttons");
lv_obj_align(label, NULL, LV_ALIGN_IN_TOP_MID, 0, 5);

/*Cria um botão normal*/
lv_obj_t * btn1 = lv_btn_create(lv_scr_act(), NULL);
lv_cont_set_fit(btn1, true, true); /*Ativa redimensionamento horizontal e vertical*/
lv_obj_align(btn1, label, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
lv_obj_set_free_num(btn1, 1);   /*Configura um único número para o botão*/
lv_btn_set_action(btn1, LV_BTN_ACTION_CLICK, btn_click_action);

/*Adiciona um rótulo para o botão*/
label = lv_label_create(btn1, NULL);
lv_label_set_text(label, "Normal");

/*Copia o botão e atribui o estado comutado. (A ação soltar é copiada também)*/
lv_obj_t * btn2 = lv_btn_create(lv_scr_act(), btn1);
lv_obj_align(btn2, btn1, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
lv_btn_set_state(btn2, LV_BTN_STATE_TGL_REL);  /*Atribui o estado comutado*/
lv_obj_set_free_num(btn2, 2);               /*Atribui um único número para o botão*/

/*Adiciona um rótulo para o botão comutado*/
label = lv_label_create(btn2, NULL);
lv_label_set_text(label, "Toggled");

/*Copia o botão e configura o estado inativo.*/
lv_obj_t * btn3 = lv_btn_create(lv_scr_act(), btn1);
lv_obj_align(btn3, btn2, LV_ALIGN_OUT_BOTTOM_MID, 0, 10);
lv_btn_set_state(btn3, LV_BTN_STATE_INA);   /*Atribui o estado inativo*/
lv_obj_set_free_num(btn3, 3);               /*Atribui um único número para o botão*/

/*Adiciona um rótulo para o botão inativo*/
label = lv_label_create(btn3, NULL);
lv_label_set_text(label, "Inactive");
```

Matrix de botões (lv_btnm)

Escrito pora v5.1

Visão geral

Os objetos de Matrizes de Botões podem mostrar **múltiplos botões** de acordo para um descritor de vetor de string, chamada *map*. Você pode especificar o mapa com `lv_btnm_set_map(btnm, my_map)`.

A **declaração de um mapa** se parece como `const char * map[] = {"btn1", "btn2", "btn3", ""}`. Nota que **o último elemento tem que ser uma string vazia**!

O primeiro caractere de uma string pode ser um **controle de caractere** para especificar alguns atributos:

- **bit 7..6** Sempre *0b10* para diferenciar o byte de controle vindo de caractere contextual
- **bit 5** Botão inativo
- **bit 4** Sem pressionamento longo para o botão
- **bit 3** Botões escondidos
- **bit 2..0** Comprimento relativo comparado para os botões na mesma coluna. [1..7]

É recomendado para especificar o **byte de controle como um número octal**. Por exemplo `"\213button"`. O número octal sempre começa com _2_ (bit 7..6) o meia parte é os atributos (bit 5..3) e o última parte é o comprimento (bit 2..0). Assim o exemplo descreve um botão escondido com 3 unidade de comprimento.

Use "\n" no mapa para fazer a **quebra de linha**: `{"btn1", "btn2", "\n", "btn3", ""}`. O comprimento do botão é recalculado em cada linha.

A `lv_btnm_set_action(btnm, btnm_action)` especifica uma ação para chamar quando um botão é liberado.

Você pode ativar as **comutação dos botões** quando eles são clicados. Assim podem somente ser um botão comutado em uma vez. O `lv_btnm_set_toggle(btnm, true, id)` ativa a comutação e configura o _id_ésimo botão para o modo comutável.

Uso do estilo

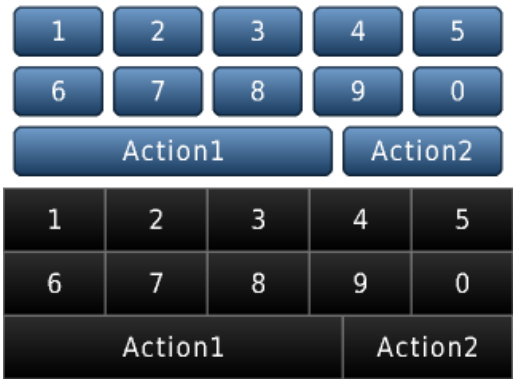
O uso da matriz de botão funciona com 6 estilos: um background e 5 estilos de botões para cada estado. Você pode configurar estilos com `lv_btnm_set_style(btn, LV_BTNM_STYLE_..., &style)`. O background e os botões usa a propriedade *style.body*. Os rótulos usam a propriedade *style.text* do estilo de botão.

- **LV_BTNM_STYLE_BG** Estilo de background. Use toda a propriedade *style.body* incluindo *padding* Padrão: _lv_style_pretty_
- **LV_BTNM_STYLE_BTN_REL** estilo do botões liberados. Padrão: _lv_style_btn_rel_
- **LV_BTNM_STYLE_BTN_PR** estilo do botões pressionados. Padrão: _lv_style_btn_pr_
- **LV_BTNM_STYLE_BTN_TGL_REL** estilo dos botões comutado-liberado. Padrão: _lv_style_btn_tgl_rel_
- **LV_BTNM_STYLE_BTN_TGL_PR** estilo dos botões comutado-liberado. Padrão: _lv_style_btn_tgl_pr_
- **LV_BTNM_STYLE_BTN_INA** estilo dos botões inativos. Padrão: _lv_style_btn_ina_

Notas

- O objeto da matriz de botão é **muito leve**. Ele cria somente a base do objeto do background e desenha os botões nele ao invés de criar muitos botões reais.

Exemplo




```c
/*Chamado quando um botão é liberado ou pressionado longamente*/
static lv_res_t btnm_action(lv_obj_t * btnm, const char *txt)
{
    printf("Button: %s released\n", txt);

    return LV_RES_OK; /*Retorna OK poque o a matriz de botão não está apagada*/
}



.
.
.

/*Cria um vetor de string dodescritor de botão*/
static const char * btnm_map[] = {"1", "2", "3", "4", "5", "\n",
                            "6", "7", "8", "9", "0", "\n",
                            "\202Action1", "Action2", ""};




/*Cria uma matriz de botão padrão*/
lv_obj_t * btnm1 = lv_btnm_create(lv_scr_act(), NULL);
lv_btnm_set_map(btnm1, btnm_map);
lv_btnm_set_action(btnm1, btnm_action);
lv_obj_set_size(btnm1, LV_HOR_RES, LV_VER_RES / 2);

/*Cira um novo estilo do background para a matriz de botão*/
static lv_style_t style_bg;
lv_style_copy(&style_bg, &lv_style_plain);
style_bg.body.main_color = LV_COLOR_SILVER;
style_bg.body.grad_color = LV_COLOR_SILVER;
style_bg.body.padding.hor = 0;
style_bg.body.padding.ver = 0;
style_bg.body.padding.inner = 0;

/*Cria 2 estilos de botões*/
static lv_style_t style_btn_rel;
static lv_style_t style_btn_pr;
lv_style_copy(&style_btn_rel, &lv_style_btn_rel);
style_btn_rel.body.main_color = LV_COLOR_MAKE(0x30, 0x30, 0x30);
style_btn_rel.body.grad_color = LV_COLOR_BLACK;
style_btn_rel.body.border.color = LV_COLOR_SILVER;
style_btn_rel.body.border.width = 1;
style_btn_rel.body.border.opa = LV_OPA_50;
style_btn_rel.body.radius = 0;

lv_style_copy(&style_btn_pr, &style_btn_rel);
style_btn_pr.body.main_color = LV_COLOR_MAKE(0x55, 0x96, 0xd8);
style_btn_pr.body.grad_color = LV_COLOR_MAKE(0x37, 0x62, 0x90);
style_btn_pr.text.color = LV_COLOR_MAKE(0xbb, 0xd5, 0xf1);

/*Cria uma segunda matriz de botão com novos estilos*/
lv_obj_t * btnm2 = lv_btnm_create(lv_scr_act(), btnm1);
lv_btnm_set_style(btnm2, LV_BTNM_STYLE_BG, &style_bg);
lv_btnm_set_style(btnm2, LV_BTNM_STYLE_BTN_REL, &style_btn_rel);
lv_btnm_set_style(btnm2, LV_BTNM_STYLE_BTN_PR, &style_btn_pr);
lv_obj_align(btnm2, btnm1, LV_ALIGN_OUT_BOTTOM_MID, 0, 0);
```

Calendário (lv_calendar)

Escrito para v5.2

Visão geral

O objeto calendário é um calendário clássico na qual pode:

- enfatizar o dia atual e semana
- enfatizar qualquer data definida pelo usuário
- mostrar o nome dos dias
- vai para o mês próximo/anterior pelo clique do botão

A atribuição e a obtenção de datas no calendário o tipo `lv_calendar_date_t` é usada na qual é uma estrutura com os campos `ano`, `mês` e `dia`.

Para configurar a **data atual** use a função `lv_calendar_set_today_date(calendar, &today_date)`.

Para configurar **a data mostrada** use `lv_calendar_set_shown_date(calendar, &shown_date)`;

A lista das **datas enfatizadas** devem ser mexido e um vetor `lv_calendar_date_t` e esse vetor pode ser passado para `lv_calendar_set_hoghlighted_dates(calendar, &highlighted_dates)`. Somente os ponteiros de vetor será salvo e então o vetor deve ser uma variável global ou estática.

O **nome dos dias** podem ser ajustados com lv_calendar_set_day_names(calendar, day_names) `onde` day_names `looks like` const char * day_names[7] = {"Su", "Mo", ...};`

Uma **ação para selecionar uma data** será suportada em `v5.3` e aogra disponível no ramo da `dev-5.3` para uso experimental.

Uso do estilo

- **LV_CALENDAR_STYLE_BG** Estilo do background usando a propriedade do `corpo` e o estilo dos dados dos números usando propriedade `texto`.
- **LV_CALENDAR_STYLE_HEADER** Estilo do cabeçalho onde o ano atual e mês são mostrados. Propriedades de `corpo` e `texto` são usados
- **LV_CALENDAR_STYLE_HEADER_PR** Estilo cabeçalho pressionado, usado quando o botáo próximo/anterior está sendo pressionado. Propriedade de `texto` são usados pelos vetores.
- **LV_CALENDAR_STYLE_DAY_NAMES** Estilo de nomes dos dias. Propriedade de `texto` são usados pelo o textos do dia e o `body.padding.ver` determina o espaço acima dos nomes do dia.
- **LV_CALENDAR_STYLE_HIGHLIGHTED_DAYS** Propriedade de `texto` são usados para ajustar o estilo dos dias enfatizados
- **LV_CALENDAR_STYLE_INACTIVE_DAYS** Propriedad de `texto` são usados para ajustar o estilo dos dias visíveis do mês próximo/anterior.
- **LV_CALENDAR_STYLE_WEEK_BOX** Propriedade de `corpo` são usadas para configurar o estilo da caixa de mês
- **LV_CALENDAR_STYLE_TODAY_BOX** Propeirdades de `corpo` e `texto` são usados para confifurar o estilo da caixa de hoje

Example


```c
/*Ceia um Objeto Calendário*/
lv_obj_t * calendar = lv_calendar_create(lv_scr_act(), NULL);
lv_obj_set_size(calendar, 240, 220);
lv_obj_align(calendar, NULL, LV_ALIGN_CENTER, 0, 0);

/*Cria um estilo para o mês atual*/
static lv_style_t style_week_box;
lv_style_copy(&style_week_box, &lv_style_plain);
style_week_box.body.border.width = 1;
style_week_box.body.border.color = LV_COLOR_HEX3(0x333);
style_week_box.body.empty = 1;
style_week_box.body.radius = LV_RADIUS_CIRCLE;
style_week_box.body.padding.ver = 3;
style_week_box.body.padding.hor = 3;

/*Cria um estilo para hoje*/
static lv_style_t style_today_box;
lv_style_copy(&style_today_box, &lv_style_plain);
style_today_box.body.border.width = 2;
style_today_box.body.border.color = LV_COLOR_NAVY;
style_today_box.body.empty = 1;
style_today_box.body.radius = LV_RADIUS_CIRCLE;
style_today_box.body.padding.ver = 3;
style_today_box.body.padding.hor = 3;
style_today_box.text.color= LV_COLOR_BLUE;

/*Cria um estilo para os dias enfatizados*/
static lv_style_t style_highlighted_day;
lv_style_copy(&style_highlighted_day, &lv_style_plain);
style_highlighted_day.body.border.width = 2;
style_highlighted_day.body.border.color = LV_COLOR_NAVY;
style_highlighted_day.body.empty = 1;
style_highlighted_day.body.radius = LV_RADIUS_CIRCLE;
style_highlighted_day.body.padding.ver = 3;
style_highlighted_day.body.padding.hor = 3;
style_highlighted_day.text.color= LV_COLOR_BLUE;

/*Aplica os estilos*/
lv_calendar_set_style(calendar, LV_CALENDAR_STYLE_WEEK_BOX, &style_week_box);
lv_calendar_set_style(calendar, LV_CALENDAR_STYLE_TODAY_BOX, &style_today_box);
lv_calendar_set_style(calendar, LV_CALENDAR_STYLE_HIGHLIGHTED_DAYS, &style_highlighted_day);


/*Configura hoje*/
lv_calendar_date_t today;
today.year = 2018;
today.month = 10;
today.day = 23;

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

/*Enfatiza alguns dias*/
static lv_calendar_date_t highlihted_days[3];       /*Somente seu ponteiro será salvo sendo assim deve ser estático*/
highlihted_days[0].year = 2018;
highlihted_days[0].month = 10;
highlihted_days[0].day = 6;

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

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

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

Gráfico (lv_chart)

Escrito para v5.1

Visão geral

Gráficos tem um background tipo retângulo com linhas de divisões horizontais e verticais. Você pode adicionar qualquer número de **séries** para os gráficos pelo `lv_chart_add_series(chart, color)`. Ele aloca dados por uma estrutura `lv_chart_series_t` na qual contem a *cor* escolhida e um vetor para os dados.

Você tem várias opções para configurar os dados de séries:

1. Configurar os valores manualmente em um vetor como `ser1->points[3] = 7` e atualizar o gráfico com `lv_chart_refresh(chart)`.
2. Usar a função `lv_chart_set_next(chart, ser, value)` para deslocar todos os dados da esquerdas e configurar um novo dado na posição à esquerda.
3. Inicializar todos os pontos para um dado valor com: `lv_chart_init_points(chart, ser, value)`.
4. Configurar todos os pontos de um vetor com: `lv_chart_set_points(chart, ser, value_array)`.

Existem quatro **tipos de display de dados**:

- LV_CHART_TYPE_NONE: não mostra os pontos. Isto pode ser usado se você gostaria de adicionar seu próprio método de desenho.
- LV_CHART_TYPE_LINE: desenha linhas entre os pontos
- LV_CHART_TYPE_COL: Desenha colunas
- LV_CHART_TYPE_POINT: Desenha pontos

Você pode especificar os tipos de display com `lv_chart_set_type(chart, TYPE)`. O tipo `LV_CHART_TYPE_LINE | LV_CHART_TYPE_POINT` é também válido para desenhar ambas linhas e pontos.

Você pode especificar os **valores das direções min. e max. em y** com `lv_chart_set_range(chart, y_min, y_max)`. O valor dos pontos serão escaladas proporcionalmente. O alcance padrão é: 0..100.

O **número de pontos** na linhas de dados podem ser modificados por `lv_chart_set_point_count(chart, point_num)`. O valor padrão é 10.

O **número das linhas de divisões horizontais e verticais** podem ser modificados por `lv_chart_set_div_line_count(chart, hdiv_num, vdiv_num)`. O padrão de configurações são linhas de divisão de 3 horizontais e 5 verticais.

Para configurar o **comprimento de linha** e **ponto do raio** use a função `lv_chart_set_series_width(chart, size)`. O padrão é: 2.

A **opacidade das linhas de dados* podem ser especificadas pelo `lv_chart_set_series_opa(chart, opa)`. O valor padrão é: OPA_COVER.

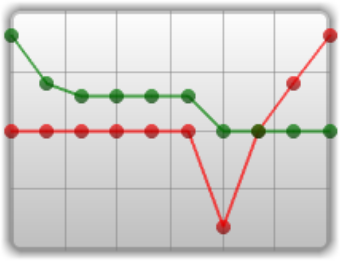
Você pode aplicar um **desaparecimento de cor escura** sobre o fundo das colunas e pontos pela função `lv_chart_set_series_darking(chart, effect)`. O padrão de nível de escuridão é OPA_50.

Uso de estilo

- **style.body** propriedade para configurar a aparência de background
- **style.line** propriedade para configurar a aparência de linhas de divisão

The series related parameters can be set directly for each chart with `lv_chart_set_series_width()`, `lv_chart_set_series_opa()` and `lv_chart_set_series_dark()`. Os parâmetros das séries relatadas pode ser configurado diretamente para cada gráfico com `lv_chart_set_series_width()`, `lv_chart_set_series_opa()` e `lv_chart_set_series_dark()`.

Exemplo



```c
/*Criar um estilo para o gráfico*/
static lv_style_t style;
lv_style_copy(&style, &lv_style_pretty);
style.body.shadow.width = 6;
style.body.shadow.color = LV_COLOR_GRAY;
style.line.color = LV_COLOR_GRAY;

/*Criar um gráfico*/
lv_obj_t * chart;
chart = lv_chart_create(lv_scr_act(), NULL);
lv_obj_set_size(chart, 200, 150);
lv_obj_set_style(chart, &style);
lv_obj_align(chart, NULL, LV_ALIGN_CENTER, 0, 0);
lv_chart_set_type(chart, LV_CHART_TYPE_POINT | LV_CHART_TYPE_LINE);   /*Mostra linhas e pontos também*/
lv_chart_set_series_opa(chart, LV_OPA_70);                            /*Opacidade da séries de dados*/
lv_chart_set_series_width(chart, 4);                                  /*Comprimento da linha e o ponto de radio*/

lv_chart_set_range(chart, 0, 100);

/*Adiciona duas séries de dados*/
lv_chart_series_t * ser1 = lv_chart_add_series(chart, LV_COLOR_RED);
lv_chart_series_t * ser2 = lv_chart_add_series(chart, LV_COLOR_GREEN);

/*Configurar o próximo pontos em 'dl1'*/
lv_chart_set_next(chart, ser1, 10);
lv_chart_set_next(chart, ser1, 50);
lv_chart_set_next(chart, ser1, 70);
lv_chart_set_next(chart, ser1, 90);

/*Diretamente configura pontos sobre 'dl2'*/
ser2->points[0] = 90;
ser2->points[1] = 70;
ser2->points[2] = 65;
ser2->points[3] = 65;
ser2->points[4] = 65;
ser2->points[5] = 65;

lv_chart_refresh(chart); /*Requerido depois da configuração direta*/
```

Check box (lv_cb)

Visão geral

Os objetos de caixa de checagem são construídos de um Botão **background** na qual contém também um Botão **bala** e um **rótulo** para realizar uma caixa de checagem clássica. O **texto** pode ser modificado pela funcão `lv_cb_set_text(cb, "New text")`.

Uma **ação** pode ser assinada por `lv_cb_set_action(cb, action)`.

Você pode manualmente **marcar / desmarcar** a checagem via `lv_cb_set_checked(cb, state)`.

Uso do estilo

O estilo da caixa de checagem pode ser modificada com `lv_cb_set_style(cb, LV_CB_STYLE_..., &style)`.

- **LV_CB_STYLE_BG** Estilo do background. Usa todas as propriedades *style.body*. O rótulo do estilo vem de *style.text*. Padrão: _lv_style_transp_
- **LV_CB_STYLE_BOX_REL** Estilo do caixa ao deixar de pressionar. Use a propriedade *style.body*. Padrão: _lv_style_btn_rel_
- **LV_CB_STYLE_BOX_PR** Estilo da caixa pressionada. Use a propriedade *style.body*. Padrão: _lv_style_btn_pr_
- **LV_CB_STYLE_BOX_TGL_REL** Estilo deixar de pressionar caixa de checagem. Use a propriedade *style.body*. Padrão: _lv_style_btn_tgl_rel_
- **LV_CB_STYLE_BOX_TGL_PR** Estilo da caixa de checagem deixar de pressionar. Use a propriedade *style.body*. Padrão: _lv_style_btn_tgl_pr_

Exemplo

```c
static lv_res_t cb_release_action(lv_obj_t * cb)
{
    /*Um caixa de checagem é clicada*/
    printf("%s state: %d\n", lv_cb_get_text(cb), lv_cb_is_checked(cb));

    return LV_RES_OK;
}

.
.
.


/**********************************************
 * Cria um container para as caixas de checagem
 **********************************************/

/*Cria o estilo da borda*/
static lv_style_t style_border;
lv_style_copy(&style_border, &lv_style_pretty_color);
style_border.glass = 1;
style_border.body.empty = 1;

/*Cria um container*/
lv_obj_t * cont;
cont = lv_cont_create(lv_scr_act(), NULL);
lv_cont_set_layout(cont, LV_LAYOUT_COL_L);      /*Organiza os filhos em uma coluna*/
lv_cont_set_fit(cont, true, true);              /*Combina o tamanho para o conteúdo*/
lv_obj_set_style(cont, &style_border);

/*****************************
 * Cria as caixas de checagem
 *****************************/

/*Cria uma caixa de checagem*/
lv_obj_t * cb;
cb = lv_cb_create(cont, NULL);
lv_cb_set_text(cb, "Potato");
lv_cb_set_action(cb, cb_release_action);

/*Copia a caixa de checagem prévia*/
cb = lv_cb_create(cont, cb);
lv_cb_set_text(cb, "Onion");

/*Copia a caixa de checagem prévia*/
cb = lv_cb_create(cont, cb);
lv_cb_set_text(cb, "Carrot");

/*Copia a caixa de checagem prévia*/
cb = lv_cb_create(cont, cb);
lv_cb_set_text(cb, "Salad");

/*Alinha o container para o meio*/
lv_obj_align(cont, NULL, LV_ALIGN_CENTER, 0, 0);
```

Container (lv_cont)

Escrito para v5.1

Visão geral

Os containeres são **objetos retangulares** com algumas características especiais.

Você pode aplicar um **layout** nos contâineres para automaticamente ordenar seus filhos. O espaçamento do layout vem das propriedades style.body.padding.hor/ver/inner. As possíveis opções de layout:

- LV_LAYOUT_OFF: Não alinha as os filhos
- LV_LAYOUT_CENTER: Alinha os filhos ao centro em colunas e mantém o espaço *pad.inner* entre eles
- LV_LAYOUT_COL_L: Alinha os filhos na coluna esquerda justificada. Mantém o espaço *pad.hor* à esquerda, espaço *pad.ver* no topo e *pad.inner* entre os filhos.
- LV_LAYOUT_COL_M: Alinha os filhos centralizados em coluna. Mantém o espaço *pad.ver* no topo e o espaço *pad.inner* entre os filhos.
- LV_LAYOUT_COL_R: Alinha os filhos na coluna justificada à direita. Mantém o espaço *pad.hor* à direita, o espaço *pad.ver* no topo e o espaço *pad.inner* entre os filhos.
- LV_LAYOUT_ROW_T: Alinha os filhos na linha justificada ao topo. Mantém o espaço *pad.hor* à esquerda, o espaço *pad.ver* ao topo e o espaço *pad.inner* entre os filhos
- LV_LAYOUT_ROW_M: Alinha os filhos em linhas centralizadas. Mantém o espaço *pad.hor* à esquerda e o espaço *pad.inner* entre os filhos.
- LV_LAYOUT_ROW_B: Alinha os filhos ao fundo da linha justificada. Mantém o espaço *pad.hor*, o espaço *pad.ver* ao fundo e o espaço *pad.inner* entre os filhos.
- LV_LAYOUT_PRETTY: Coloca muitos objetos possíveis em uma linha (com pelo menos o espaço *pad.inner* e o espaço *pad.hor* ao lado) e começa uma nova linha. Divide o espaço em cada linha igualmente entre os filhos. Mantém o espaço *pad.ver* no topo e o espaço *pad.inner* entre as linhas.
- LV_LAYOUT_GRID: Similar ao PRETTY LAYOUT mas não divide o espaço horizontal igualmente apenas deixa o espaço *pad.hor*

Você pode ativar um método **auto ajuste** na qual automaticamente configura o tamanho do contâiner para incluir todos os filhos. Ele manterá o espaço *pad.hor* à esquerda e à direita e o espaço *pad.ver* no topo. O auto ajuste pode ativar horizontalmente, verticalmente o em ambas as direções com a função `lv_cont_set_fit(cont, true, true)`. O segundo parâmetro é a horizontal, o terceiro parâmetro é a ativação do ajuste vertical.

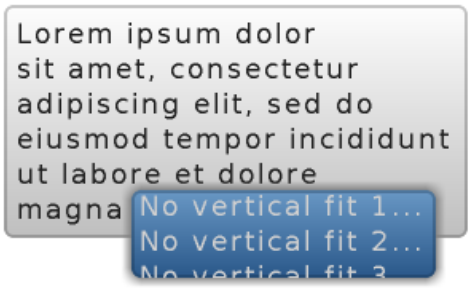
Uso do estilo

- Propriedades **style.body** são usadas

Notas

- Você **não pode atribuir a posição do filho com os ajustes hor/ver ativados**. Vamos imaginar o que acontece. Se você mudar a posição de somente um filho quando o ajuste estiver ativado no contâiner irá mover/ajustar "em torno" da nova posição. Se você tem mais objetos em um contâiner então você pode alinhar eles relativamente a cada outro. Com uma solução alternativa você pode criar um pequeno objeto transparente no contâiner. Ele irá fixar o contâiner para não "seguir" os filhos.

Exemplo




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

/*Adiciona um texto ao contâiner*/
lv_obj_t * txt = lv_label_create(box1, NULL);
lv_label_set_text(txt, "Lorem ipsum dolor\n"
                       "sit amet, consectetur\n"
                       "adipiscing elit, sed do\n"
                       "eiusmod tempor incididunt\n"
                       "ut labore et dolore\n"
                       "magna aliqua.");

lv_obj_align(box1, NULL, LV_ALIGN_IN_TOP_LEFT, 10, 10);      /*Alinha o contâiner*/

/*Cria um estilo*/
static lv_style_t style;
lv_style_copy(&style, &lv_style_pretty_color);
style.body.shadow.width = 6;
style.body.padding.hor = 5;                                  /*Configura um espaçamento grande*/

/*Cria um outro contâiner*/
lv_obj_t * box2;
box2 = lv_cont_create(lv_scr_act(), NULL);
lv_obj_set_style(box2, &style);     /*Configura o novo estilo*/
lv_cont_set_fit(box2, true, false); /*Não ativa o ajuste vertical*/
lv_obj_set_height(box2, 55);        /*Configura uma altura fixa*/

/*Adiciona um texto ao novo contâiner*/
lv_obj_t * txt2 = lv_label_create(box2, NULL);
lv_label_set_text(txt2, "No vertical fit 1...\n"
                        "No vertical fit 2...\n"
                        "No vertical fit 3...\n"
                        "No vertical fit 4...");

/*Alinha o contâiner para o fundo do anterior*/
lv_obj_align(box2, box1, LV_ALIGN_OUT_BOTTOM_MID, 30, -30);
```

Lista suspensa (lv_ddlist)

Escrito para v5.3, revisão 2

Visão geral

Lista suspensa permite a você simplesmente **selecionar uma opção entre várias**. A lista suspensa é fechada por padrão e mostra o texto selecionado corrente. Se você clicar nele nessa lista aberta e todas as opções são mostradas.

As **opções** que são passadas à lista suspensa como um **string** com `lv_ddlist_set_options(ddlist, options)`. As opções devem ser separadas por `\n`. Por exemplo: `"First\nSecond\nThird"`.

Você pode **selecionar uma opção manualmente** com `lv_ddlist_set_selected(ddlist, id)`, onde _id_ é o índice de uma opção.

Uma **função callback** pode ser especificada com `lv_ddlist_set_action(ddlist, my_action)` para chamar quando uma nova opção é selecionada.

Por padrão a **altura** das listas é ajustada automaticamente para mostrar todas as opções. O `lv_ddlist_set_fix_height(ddlist, h)` configura a altura fixa para a lista aberta.

O **comprimento** é também ajustado automaticamente. Para evitar isto aplique `lv_ddlist_set_hor_fit(ddlist, false)` e configure o comprimento manualmente através de `lv_obj_set_width(ddlist, width)`.

Similarmente para Pagína com altura fixada a lista suspensa que suporta vários **modos de barra de rolagem de display**. Ele pode ser configurado por `lv_ddlist_set_sb_mode(ddlist, LV_SB_MODE_...)`.

A animação da lista suspensa do tempo abre/fecha é ajustado por `lv_ddlist_set_anim_time(ddlist, anim_time)`. Tempo zero de animação significa nenhuma animação.

Novo em v5.3 é a abilidade para ativar um vertor no lado da lista suspensa. Para usar esse recurso você pode chamar `lv_ddlist_set_draw_arrow(ddlist, true)`.

Uso do estilo

O `lv_ddlist_set_style(ddlist, LV_DDLIST_STYLE_..., &style)` configura os estilos de uma lista suspensa.

- **LV_DDLIST_STYLE_BG** Estilo do background. Toda propriedade *style.body* são usadas. Ele é usado para o estilo do rótulo do *style.text*. Padrão: _lv_style_pretty_
- **LV_DDLIST_STYLE_SEL** Estilo da opção selecionada. A propriedade *style.body* é usada. A opção selecionada será recolorida com *text.color*. Padrão: _lv_style_plain_color_
- **LV_DDLIST_STYLE_SB** Estilo da barra de rolagem. A propriedade *style.body* é usada. Padrão: _lv_style_plain_color_

Exemplo

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

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

    return LV_RES_OK; /*Retorna OK se a lista suspensa não foi apagada*/
}


.
.
.

/*Cria uma lista suspensa*/
lv_obj_t * ddl1 = lv_ddlist_create(lv_scr_act(), NULL);
lv_ddlist_set_options(ddl1, "Apple\n"
                            "Banana\n"
                            "Orange\n"
                            "Melon\n"
                            "Grape\n"
                            "Raspberry");
lv_obj_align(ddl1, NULL, LV_ALIGN_IN_TOP_LEFT, 30, 10);
lv_obj_set_free_num(ddl1, 1);               /*Atribui uma única ID*/
lv_ddlist_set_action(ddl1, ddlist_action);  /*Atribui uma função para chamar quando uma nova opção é escolhida*/

/*Cria um estilo*/
static lv_style_t style_bg;
lv_style_copy(&style_bg, &lv_style_pretty);
style_bg.body.shadow.width = 4; /*Ativa a sombra*/
style_bg.text.color = LV_COLOR_MAKE(0x10, 0x20, 0x50);

/*Copia a lista suspensa e configura o novo style_bg*/
lv_obj_t * ddl2 = lv_ddlist_create(lv_scr_act(), ddl1);
lv_obj_align(ddl2, NULL, LV_ALIGN_IN_TOP_RIGHT, -30, 10);
lv_obj_set_free_num(ddl2, 2);       /*Atribui uma única ID*/
lv_obj_set_style(ddl2, &style_bg);
```

Medidor/Indicador (lv_gauge)

Escrito para v5.1

Visão geral

O indicador é um medidor com **rótulos de escala** e **agulhas**. Você pode usar a função `lv_gauge_set_scale(gauge, angle, line_num, label_cnt)` para ajustar o ânguloa da escala e o número das linhas de escalas e rótulos. As configurações são: 220 graus, 6 rótulos de escala e 21 linhas.

O indicador pode mostrar **mais do que uma agulha**. Use a função `lv_gauge_set_needle_count(gauge, needle_num, color_array)` para configurar o número de agulhas e um vetor com cores para cada agulha. (O vetor precisa ser estático ou variável global).

Você pode usar `lv_gauge_set_value(gauge, needle_id, value)` para **configurar o valor da agulha**.

Para configurar um **valor crítico** use `lv_gauge_set_critical_value(gauge, value)`. A cor de escala será mudada para *line.color* depois desse valor. (padrão: 80)

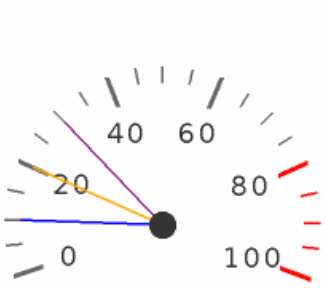
O **alcance** do medidor pode ser especificado por `lv_gauge_set_range(gauge, min, max)`.

Uso do estilo

O medidor usa um estilo na qual pode ser configurado por `lv_gauge_set_style(gauge, &style)`. As propriedades do medidor são derivadas dos atributos dos seguintes estilos:

- **body.main_color** cor da linha no início da escala
- **body.grad_color** cor da linha no fim da escala (gradiente com a cor principal)
- **body.padding.hor** comprimento da linha
- **body.padding.inner** distância do rótulo da linhas de escala
- **line.width** comprimento da linha
- **line.color** cor da linha depois do valor crítico
- **text.font/color/letter_space** atributos do rótulo

Exemplo



```c
/*Cria um estilo*/
static lv_style_t style;
lv_style_copy(&style, &lv_style_pretty_color);
style.body.main_color = LV_COLOR_HEX3(0x666);     /*Cor da linha no início*/
style.body.grad_color =  LV_COLOR_HEX3(0x666);    /*Cor da linha no fim*/
style.body.padding.hor = 10;                      /*Coomprimento da escala da linha*/
style.body.padding.inner = 8 ;                    /*Escala do espaçamento do rótulo*/
style.body.border.color = LV_COLOR_HEX3(0x333);   /*Cor do círculo do meio da agulha*/
style.line.width = 3;
style.text.color = LV_COLOR_HEX3(0x333);
style.line.color = LV_COLOR_RED;                  /*Cor da linha depois do valor crítico*/


/*Descreve a cor para as agulhas*/
static lv_color_t needle_colors[] = {LV_COLOR_BLUE, LV_COLOR_ORANGE, LV_COLOR_PURPLE};

/*Cria um medidor*/
lv_obj_t * gauge1 = lv_gauge_create(lv_scr_act(), NULL);
lv_gauge_set_style(gauge1, &style);
lv_gauge_set_needle_count(gauge1, 3, needle_colors);
lv_obj_align(gauge1, NULL, LV_ALIGN_CENTER, 0, 20);

/*Configura os valores*/
lv_gauge_set_value(gauge1, 0, 10);
lv_gauge_set_value(gauge1, 1, 20);
lv_gauge_set_value(gauge1, 2, 30);
```

Imagem (lv_img)

Escrito para v5.1

Visão geral

As imagens são os objetos básicos para **mostrar imagens**. Para promover máxima flexibilidade da **origem da imagem** você pode:

- uma variável no código (uma matriz C com pixels)
- um arquivo armazenado externamente (como um SD card)
- um texto com Símbolos

Para configurar a origem de uma imagem a função `lv_img_set_src` pode ser usada.

Para gerar um vetor de pixel de uma imagem **de um PNG, JPG ou BMP** use o Conversor de imagem online e personalize a imagem convertida com o seu ponteiro:

```
lv_img_set_src(img1, &converted_img_var);
```

Para usar **arquivos externos** você precisa converter os arquivos de imagem usando a ferramenta de conversão online mas agora você deve selecionar o formato da saída binária. Para ver como carregar a imagem externa do LittlevGL veja o [Tutorial] (https://github.com/littlevgl/lv_examples/tree/master/lv_tutorial/6_images).

You can set a **symbol** from lv_symbol_def.h too. In this case, the image will be rendered as text according to the **font** specified in the style. It enables to use light weighted mono-color Neste caso, a imagem será renderizada como texto de acordo com a **fonte** especificada no estilo. Ele ativa o uso leve de "letras" monocromáticas ao invés de imagens reais. Você pode atribuir símbolos como este:

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

A (variável) interna e as imagens externas suporta 2 métodos de **carregadores de transparencia**:

- **Croma key** `LV_COLOR_TRANSP` (lv_conf.h) será transparente
- **Byte alfa** Adiciona um byte alfa para cada pixel

Essas opções podem ser selecionadas no conversor online de fonte.

As imagens podem ser **recolorizadas em tempo real de execução** para qualquer cor de acordo com o brilho dos pixels. Isto é muito útil para mostrar diferentes estados (selecionado, inativo, pressionado, etc) de uma uma imagem sem guardar mais versões da mesma imagem. Esta característica pode ser ativada no estilo ao atribuir `img.intense` entre `LV_OPA_TRANSP` (nenhuma cor, valor: 0) e `LV_OPA_COVER` (total recolorização, valor: 255). O padrão é `LV_OPA_TRANSP` então assim esta esta atribuição é desativada.

É possível **automaticamente atribuir o tamanho** do comprimento e a altura da origem do objeto da imagem aplicando a função `lv_img_set_auto_size(image, true)`. Se *auto size* estiver ativado, então um novo arquivo é atribuido ao tamanho do objeto é automaticamente mudado. Posteriormente você poderá mudar o tamanho manualmente. Se o objeto é maior do que o tamanho da imagem em qualquer direção entãoo a imagem será repetida como um mosaico. O tamanho automático é ativado por padrão se a imagem não é uma tela.

Os estilos das imagens padrões é NULL assim elas **herdam o estilo do pai**.

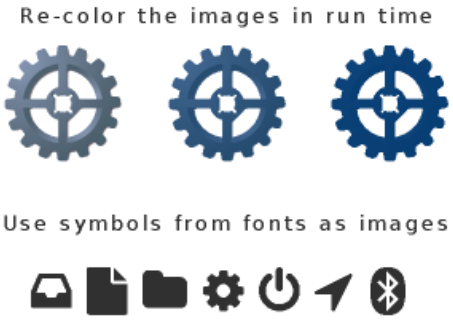
Uso do estilo

- Para imagens **style.img**
- Para símbolos **style.text**

Notas

- Nomes de símbolos (como SYMBOL_EDIT) são strings curtas, contudo, você pode concatenar eles para mostrar mais símbolos

Exemplo




```c
/*Declara uma uma imagem roda dentada*/
LV_IMG_DECLARE(img_cw);

[...]

/************************************
 * Cria três imagens e recoloriza elas
 ************************************/

/*Cria a primeira imagem sem recolorizar*/
lv_obj_t * img1 = lv_img_create(lv_scr_act(), NULL);
lv_img_set_src(img1, &img_cw);
lv_obj_align(img1, NULL, LV_ALIGN_IN_TOP_LEFT, 20, 40);

/*Cria estilo para recolorizar com a cor azul claro*/
static lv_style_t style_img2;
lv_style_copy( &style_img2, &lv_style_plain);
style_img2.image.color = LV_COLOR_HEX(0x003b75);
style_img2.image.intense = LV_OPA_50;

/*Cria uma imagem com o estilo azul claro*/
lv_obj_t * img2 = lv_img_create(lv_scr_act(), img1);
lv_obj_set_style(img2, &style_img2);
lv_obj_align(img2, NULL, LV_ALIGN_IN_TOP_MID, 0, 40);

/*Cria estilo para recolorizar com a cor azul escuro*/
static lv_style_t style_img3;
lv_style_copy(&style_img3, &lv_style_plain);
style_img3.image.color = LV_COLOR_HEX(0x003b75);
style_img3.image.intense = LV_OPA_90;

/*Cria uma imagem com o estilo azul escuro*/
lv_obj_t * img3 = lv_img_create(lv_scr_act(), img2);
lv_obj_set_style(img3, &style_img3);
lv_obj_align(img3, NULL, LV_ALIGN_IN_TOP_RIGHT, -20, 40);


/************************************
 * Cria uma imagem com símbolos
 ************************************/

/*Cria uma string de símbolos*/
char buf[32];
sprintf(buf, "%s%s%s%s%s%s%s",
             SYMBOL_DRIVE, SYMBOL_FILE, SYMBOL_DIRECTORY, SYMBOL_SETTINGS,
             SYMBOL_POWER, SYMBOL_GPS, SYMBOL_BLUETOOTH);

/*Cria estilo com uma fonte de símbolo*/
static lv_style_t style_sym;
lv_style_copy(&style_sym, &lv_style_plain);
// As fontes embutidas são extendidas com símbolos
style_sym.text.font = &lv_font_dejavu_60;
style_sym.text.letter_space = 10;

/*Cria uma imagem e usa a string como origem*/
lv_obj_t * img_sym = lv_img_create(lv_scr_act(), NULL);
lv_img_set_src(img_sym, buf);
lv_img_set_style(img_sym, &style_sym);
lv_obj_align(img_sym, NULL, LV_ALIGN_IN_BOTTOM_MID, 0, -30);


/*Cria rótulos de descrição*/
lv_obj_t * label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(label, "Re-color the images in run time");
lv_obj_align(label, NULL, LV_ALIGN_IN_TOP_MID, 0, 15);

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

Botão de imagem (lv_imgbtn)

Escrito para v5.2

Visão geral

No botão de imagem é muito similar ao objeto Botão. A única diferença é que seu visual é definido pela imagem que o usuário usa em cada estado ao invés de desenhar um botão. Antes de ler isto, por favorr, aprenda como os Botões funcionam no LittelvGL: link para o botão

Para configurar a imagem em um estado a `lv_imgbtn_set_src(imgbtn, LV_BTN_STATE_..., &img_src)` A origem da imagem funciona da mesma forma como descrito no objeto Imagem TODO link

Similarmente ao objeto Botão **as ações (callbacks) podem ser assinadas** pelo `lv_imgbtn_set_action(imgbtn, LV_BTN_ACTION_..., my_action)`.

Os **estados** também funcionam com objeto Botão. Ele pode ser configurado com `lv_imgbtn_set_state(imgbtn, LV_BTN_STATE_...)`.

Os recursod de **comutações** podem ser ativadas com `lv_imgbtn_set_toggle(imgbtn, true)`

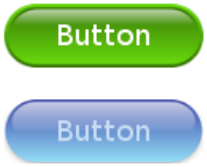
Uso do estilo

Os Botõees de imagem podem ter estilos únicos para cada estilo. Todos as propriedades `style.image` usada pelo botão de imagem:

- **image.color** Recoloriza a imagem para essa cor de acordo com `intense`
- **image.intense** A extensão da recolorização (0..255 or `LV_OPA_0/10/20..100`)
- **image.opa** A opacidade do objeto (0..255 or `LV_OPA_0/10/20..100`)

Notas

Exemplo



```
/*Cria estilo para fazer o botão mais escuro quando pressionado*/
lv_style_t style_pr;
lv_style_copy(&style_pr, &lv_style_plain);
style_pr.image.color = LV_COLOR_BLACK;
style_pr.image.intense = LV_OPA_50;
style_pr.text.color = LV_COLOR_HEX3(0xaaa);

LV_IMG_DECLARE(imgbtn_green);
LV_IMG_DECLARE(imgbtn_blue);

/*Cria um botão de imagem*/
lv_obj_t * imgbtn1 = lv_imgbtn_create(lv_scr_act(), NULL);
lv_imgbtn_set_src(imgbtn1, LV_BTN_STATE_REL, &imgbtn_green);
lv_imgbtn_set_src(imgbtn1, LV_BTN_STATE_PR, &imgbtn_green);
lv_imgbtn_set_src(imgbtn1, LV_BTN_STATE_TGL_REL, &imgbtn_blue);
lv_imgbtn_set_src(imgbtn1, LV_BTN_STATE_TGL_PR, &imgbtn_blue);
lv_imgbtn_set_style(imgbtn1, LV_BTN_STATE_PR, &style_pr);        /*Usa o estilo mais escuro no estado pressionado*/
lv_imgbtn_set_style(imgbtn1, LV_BTN_STATE_TGL_PR, &style_pr);
lv_imgbtn_set_toggle(imgbtn1, true);
lv_obj_align(imgbtn1, NULL, LV_ALIGN_CENTER, 0, -40);

/*Cria um rótulo no botão de imagem*/
lv_obj_t * label = lv_label_create(imgbtn1, NULL);
lv_label_set_text(label, "Button");

/*Copia a primeira imagem do botão e configura modo comutável*/
lv_obj_t * imgbtn2 = lv_imgbtn_create(lv_scr_act(), imgbtn1);
lv_btn_set_state(imgbtn2, LV_BTN_STATE_TGL_REL);
lv_obj_align(imgbtn2, imgbtn1, LV_ALIGN_OUT_BOTTOM_MID, 0, 20);

/*Cria um rótulo no botão de imagem*/
label = lv_label_create(imgbtn2, NULL);
lv_label_set_text(label, "Button");
```

Teclado (lv_kb)

Escrito para v5.1

Visão geral

Como o nome mostra o objeto **Teclado** provém um teclado para **escrever texto**. Você pode assinar uma Área de texto para o Teclado para colocar os caracteres digitados lá. Para assinar a Área de texto use `lv_kb_set_ta(kb, ta)`.

O Teclado contém um botão _Ok_ e um botão *Hide*. Em ações ok e hide você podem ser especificados por `lv_kb_set_ok_action(kb, action)` e `lv_kb_set_hide_action(kb, action)` para adicionar callbacks para os cliques Ok/Hide. Se nenhuma ação é especificada então os botões irá apagar o Teclado.

A Área de texto assinada **cursor** pode ser **gerenciado** teclado: quando o teclado é assinado a área de texto prévio do cursor será escondido e a nova área de texto será mostrada. Clicando em _Ok_ ou *Hide* irá também esconder o cursor. O gerenciador do cursor é ativado pelo `lv_kb_set_cursor_manage(kb, true)`. O padrão é não gerenciar.

Os teclados tem dois **modos**:

- LV_KB_MODE_TEXT: mostra letras, números e caracteres especiais
- LV_KB_MODE_NUM: mostra números, sinais +/- e pontos

Para configurar o modo use `lv_kb_set_mode(kb, mode)`. O padrão é _LV_KB_MODE_TEXT_

Você pode especificar um **novo mapa** (layout) para o teclado com `lv_kb_set_map(kb, map)`. Ele funciona como a Matriz de botões assim, controlar caractere pode ser adicionado ao layout para configurar o comprimento do botão e outros atributos. Mantenha em mente que usar as seguintes palavras-chave terá o mesmo efeito com o mapeamento original: _SYMBOL_OK_, _SYMBOL_CLOSE_, _SYMBOL_LEFT_, _SYMBOL_RIGHT_, *ABC*, *abc*, *Enter*, *Del*, _#1_, +/- .

Uso do estilo

O teclado funciona com 6 estilos: um background e 5 estilos de botões para cada estado. Você pode confiigurar estilos com `lv_kb_set_style(btn, LV_KB_STYLE_..., &style)`. O background e os botões use a propriedade *style.body*. Para os rótulos use a propriedade *style.text* dos estilos de botões.

- **LV_KB_STYLE_BG** Estilo do background. Use todas as propriedades *style.body* incluindo *padding*. Padrão: _lv_style_pretty_
- **LV_KB_STYLE_BTN_REL** estilo ao deixar de pressionar. Padrão: _lv_style_btn_rel_
- **LV_KB_STYLE_BTN_PR** estilo ao pressionar. Padrão: _lv_style_btn_pr_
- **LV_KB_STYLE_BTN_TGL_REL** estilo ao comutar. Padrão: _lv_style_btn_tgl_rel_
- **LV_KB_STYLE_BTN_TGL_PR** estilo do botão comutado. Padrão: _lv_style_btn_tgl_pr_
- **LV_KB_STYLE_BTN_INA** estilo do botão inativo. Padrão: _lv_style_btn_ina_

Exemplo

```
/*Cria estilos para o teclado*/
static lv_style_t rel_style, pr_style;

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

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

/*Cria um teclado e aplica os estilos*/
lv_obj_t *kb = lv_kb_create(lv_scr_act(), NULL);
lv_kb_set_cursor_manage(kb, true);
lv_kb_set_style(kb, LV_KB_STYLE_BG, &lv_style_transp_tight);
lv_kb_set_style(kb, LV_KB_STYLE_BTN_REL, &rel_style);
lv_kb_set_style(kb, LV_KB_STYLE_BTN_PR, &pr_style);

/*Cria uma área de texto. O teclado escreverá aqui*/
lv_obj_t *ta = lv_ta_create(lv_scr_act(), NULL);
lv_obj_align(ta, NULL, LV_ALIGN_IN_TOP_MID, 0, 10);
lv_ta_set_text(ta, "");

/*Assina a área de texto para o teclado*/
lv_kb_set_ta(kb, ta);
```

Lista (lv_list)

Escrito para v5.1

Visão geral

As listas são construídas de uma **Página** e **Botões** como background. Os Botões contém uma uma imagem opcional (na qual pode ser um símbolo também) e um Rótulo. Quando a lista se torna longa o bastante ela pode ser deslizável. O **comprimento dos botões** é configurado para o máximo de acordo com o comprimento do objeto. A **altura** dos botões sãoo ajustados automaticamente de acordo ao conteúdoo (altura do conteúdo + style.body.padding.ver).

Você pode **adicionar novos elementos na lista** com `lv_list_add(list, "U:/img", "Text", rel_action)` ou com icone de símbolo `lv_list_add(list, SYMBOL_EDIT, "Edit text")`. Se você não quer adicionar imagem use `""` como nome de arquivo. A função retorna com um ponteiro ao botão criado para permitir configurações futuras

Você pode usar `lv_list_get_btn_label(list_btn)` e `lv_list_get_btn_img(list_btn)` para **obter o rótulo e a imagem** de uma lista de botão.

Na ação de lançamento de um botãoo você pode obter o **texto do botão** com `lv_list_get_btn_text(button)`. Isto ajuda a identificar a lista do elemento lançada.

Para **apagar a lista de elemento** use `lv_obj_del()` no valor de retorno de `lv_list_add()`.

Você pode **navegar manualmente** na lista com `lv_list_up(list)` e `lv_list_down(list)`.

Você pode focar em um botão diretamente usando `lv_list_focus(btn, anim_en)`.

O **tempo de animação** de movimentos acima/abaixo/foco pode ser configurado via: `lv_list_set_anim_time(list, anim_time)`. Zero animação significa nenhuma animação.

Uso do estilo

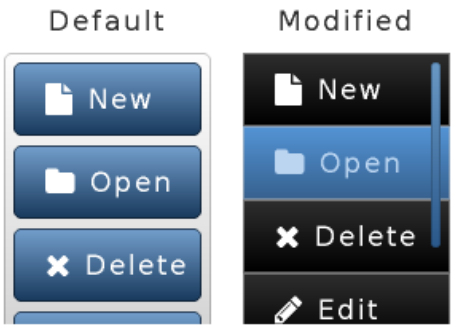
A função `lv_list_set_style(list, LV_LIST_STYLE_..., &style)` configura o estilo de uma lista. Para explicação dos detalhes de _BG_, *SCRL* e _SB_ veja Página

- **LV_LIST_STYLE_BG** lisa o estilo do background. Padrão: _lv_style_transp_fit_
- **LV_LIST_STYLE_SCRL** parte deslizável do estilo. Padrão: _lv_style_pretty_
- **LV_LIST_STYLE_SB** estilo da barra de rolagem. Padrão: _lv_style_pretty_color_
- **LV_LIST_STYLE_BTN_REL** estilo do botão deixado de pressionar. Padrão: _lv_style_btn_rel_
- **LV_LIST_STYLE_BTN_PR** estilo do botão pressionado. Padrão: _lv_style_btn_pr_
- **LV_LIST_STYLE_BTN_TGL_REL** estilo do botão comutado. Padrão _lv_style_btn_tgl_rel_
- **LV_LIST_STYLE_BTN_TGL_PR** estilo do botão pressionado. Padrão: _lv_style_btn_tgl_pr_
- **LV_LIST_STYLE_BTN_INA** estilo do botão inativo. Padrão: _lv_style_btn_ina_

Notas

- Você pode configurar um background transparente para a lista. Neste caso, se você tem somente umas poucas listas de botões a lista se parecerar menor mas se torna deslizável quanto mais lista de elementos são adicionadas.
- O modo do comprimento padrão dos rótulos de botão é `LV_LABEL_LONG_ROLL`. Você pode modificá-lo manualmente. Use `lv_list_get_btn_label()` para obter rótulos de botões.
- Para **modificar a altura dos botões** ajuste o campo *body.padding.ver* do estilo correspondente (LV_LIST_STYLE_BTN_REL , LV_LIST_STYLE_BTN_PR etc.)

Exemplo


Default
Modified
New
Open
Delete
New
Open
Delete
Edit

```c
/*Serpa chamado no clique na lista de botão*/
static lv_res_t list_release_action(lv_obj_t * list_btn)
{
    printf("List element click:%s\n", lv_list_get_btn_text(list_btn));

    return LV_RES_OK; /*Retorna OK porque a lista não foi apagada*/
}

.
.
.

/************************
 * Cria a lista padrão
 ************************/

/*Cria a lista*/
lv_obj_t * list1 = lv_list_create(lv_scr_act(), NULL);
lv_obj_set_size(list1, 130, 170);
lv_obj_align(list1, NULL, LV_ALIGN_IN_TOP_LEFT, 20, 40);

/*Adiciona a lista de elementos*/
lv_list_add(list1, SYMBOL_FILE, "New", list_release_action);
lv_list_add(list1, SYMBOL_DIRECTORY, "Open", list_release_action);
lv_list_add(list1, SYMBOL_CLOSE, "Delete", list_release_action);
lv_list_add(list1, SYMBOL_EDIT, "Edit", list_release_action);
lv_list_add(list1, SYMBOL_SAVE, "Save", list_release_action);

/*Cria um rótulo acima da lista*/
lv_obj_t * label;
label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(label, "Default");
lv_obj_align(label, list1, LV_ALIGN_OUT_TOP_MID, 0, -10);

/*********************
 * Cria novos estilos
 *********************/
/*Cria um estilo de barra de rolagem*/
static lv_style_t style_sb;
lv_style_copy(&style_sb, &lv_style_plain);
style_sb.body.main_color = LV_COLOR_BLACK;
style_sb.body.grad_color = LV_COLOR_BLACK;
style_sb.body.border.color = LV_COLOR_WHITE;
style_sb.body.border.width = 1;
style_sb.body.border.opa = LV_OPA_70;
style_sb.body.radius = LV_RADIUS_CIRCLE;
style_sb.body.opa = LV_OPA_60;

/*Cria estilos para os botões*/
static lv_style_t style_btn_rel;
static lv_style_t style_btn_pr;
lv_style_copy(&style_btn_rel, &lv_style_btn_rel);
style_btn_rel.body.main_color = LV_COLOR_MAKE(0x30, 0x30, 0x30);
style_btn_rel.body.grad_color = LV_COLOR_BLACK;
style_btn_rel.body.border.color = LV_COLOR_SILVER;
style_btn_rel.body.border.width = 1;
style_btn_rel.body.border.opa = LV_OPA_50;
style_btn_rel.body.radius = 0;

lv_style_copy(&style_btn_pr, &style_btn_rel);
style_btn_pr.body.main_color = LV_COLOR_MAKE(0x55, 0x96, 0xd8);
style_btn_pr.body.grad_color = LV_COLOR_MAKE(0x37, 0x62, 0x90);
style_btn_pr.text.color = LV_COLOR_MAKE(0xbb, 0xd5, 0xf1);

/***************************************
 * Cria uma lista com estilos modificados
 ***************************************/

/*Copia a lista prévia*/
lv_obj_t * list2 = lv_list_create(lv_scr_act(),list1);
lv_obj_align(list2, NULL, LV_ALIGN_IN_TOP_RIGHT, -20, 40);
lv_list_set_sb_mode(list2, LV_SB_MODE_AUTO);
lv_list_set_style(list2, LV_LIST_STYLE_BG, &lv_style_transp_tight);
lv_list_set_style(list2, LV_LIST_STYLE_SCRL, &lv_style_transp_tight);
lv_list_set_style(list2, LV_LIST_STYLE_BTN_REL, &style_btn_rel); /*Configura o estilo do novo botão*/
lv_list_set_style(list2, LV_LIST_STYLE_BTN_PR, &style_btn_pr);

/*Cria um rótulo acima da lista*/
label = lv_label_create(lv_scr_act(), label);       /*Copia o rótulo anterior*/
lv_label_set_text(label, "Modified");
lv_obj_align(label, list2, LV_ALIGN_OUT_TOP_MID, 0, -10);
```

LED (lv_led)

Escrito para v5.1

Visão geral

Os LEDs são objetos retângulos (ou círculos). Você pode atribuir seu **brilho** com `lv_led_set_bright(led, bright)`. O brilho deve ser entre 0 (mais escuro) e 255 (mais claro).

Use `lv_led_on(led)` e `lv_led_off(led)` para configurar o brilho para um valor ON ou OFF pré-definido. O `lv_led_toggle(led)` comuta entre os estados ON e OFF.

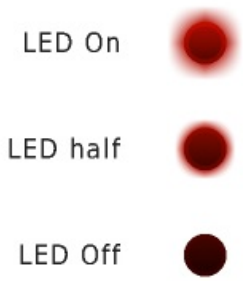
Uso do estilo

O LED usa um dos estilos na qual pode ser configurado por `lv_led_set_style(led, &style)`. Para determinar a aparência a propriedade **style.body** são usadas. As cores são mais escuras e mostra o comprimento que é reduzido em um menor brilho e ganho do seu valor originall emm 255 para mostrar um efeito luminoso. O estilo padrão é `lv_style_pretty_color`.

Notas

- Tipicamente o estilo padrão não está disponível, contudo você pode criar seu próprio estilo. Veja os exemplos.

Exemplo



```c
/*Cria um estilo para o LED*/
static lv_style_t style_led;
lv_style_copy(&style_led, &lv_style_pretty_color);
style_led.body.radius = LV_RADIUS_CIRCLE;
style_led.body.main_color = LV_COLOR_MAKE(0xb5, 0x0f, 0x04);
style_led.body.grad_color = LV_COLOR_MAKE(0x50, 0x07, 0x02);
style_led.body.border.color = LV_COLOR_MAKE(0xfa, 0x0f, 0x00);
style_led.body.border.width = 3;
style_led.body.border.opa = LV_OPA_30;
style_led.body.shadow.color = LV_COLOR_MAKE(0xb5, 0x0f, 0x04);
style_led.body.shadow.width = 10;


/*Cria um LED e liga o LED*/
lv_obj_t * led1  = lv_led_create(lv_scr_act(), NULL);
lv_obj_set_style(led1, &style_led);
lv_obj_align(led1, NULL, LV_ALIGN_IN_TOP_MID, 40, 40);
lv_led_on(led1);

/*Copia o LED anterior e configura seu brilho*/
lv_obj_t * led2  = lv_led_create(lv_scr_act(), led1);
lv_obj_align(led2, led1, LV_ALIGN_OUT_BOTTOM_MID, 0, 40);
lv_led_set_bright(led2, 190);

/*Copia o LED anterior e desliga o LED*/
lv_obj_t * led3  = lv_led_create(lv_scr_act(), led1);
lv_obj_align(led3, led2, LV_ALIGN_OUT_BOTTOM_MID, 0, 40);
lv_led_off(led3);

/*Cria 3 rótulos próximo ao LEDs*/
lv_obj_t * label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(label, "LED On");
lv_obj_align(label, led1, LV_ALIGN_OUT_LEFT_MID, -40, 0);

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

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

Linha (lv_line)

Escrito para v5.1

Visão geral

O objeto linha é capaz de **desenhar linhas retas** entre um conjunto de pontos. Os pontos tem que ser guardados em um vetor `lv_point_t` e passado ao objeto pela função `lv_line_set_points(lines, point_array, point_num)`.

É possível **automaticamente configurar o tamanho** do objeto linha de acordo com seus pontos. Você pode ativá-lo com a função `lv_line_set_auto_size(line, true)`. Se atiado quando os pontos estão configurados então o comprimento do objeto e a altura irão ser mudados de acordo com as coordenadas máximas x e y entre os pontos. O *auto size* está ativado por padrão.

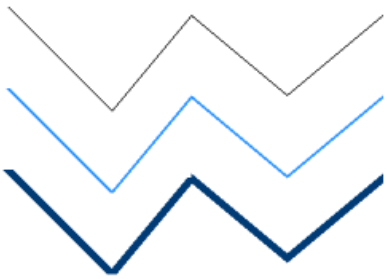
Basicamente o ponto *y == 0* está no topo do objeto mas você pode **innverter a coordenada y** com `lv_line_set_y_invert(line, true)`. Depois disso as coordenadas serão subtraídas da altura do objeto.

Uso do estilo

- Propriedades **style.line** são usadas

Notas

Exemplo



```
/*Cria um vetor para os pontos da linha*/
static lv_point_t line_points[] = { {5, 5}, {70, 70}, {120, 10}, {180, 60}, {240, 10} };

/*Cria linha com o estilo padrão*/
lv_obj_t * line1;
line1 = lv_line_create(lv_scr_act(), NULL);
lv_line_set_points(line1, line_points, 5);     /*Set the points*/
lv_obj_align(line1, NULL, LV_ALIGN_IN_TOP_MID, 0, 20);

/*Cria novo estilo (azul claro fino)*/
static lv_style_t style_line2;
lv_style_copy(&style_line2, &lv_style_plain);
style_line2.line.color = LV_COLOR_MAKE(0x2e, 0x96, 0xff);
style_line2.line.width = 2;

/*Copia a linha anterior e aplica um novo estilo*/
lv_obj_t * line2 = lv_line_create(lv_scr_act(), line1);
lv_line_set_style(line2, &style_line2);
lv_obj_align(line2, line1, LV_ALIGN_OUT_BOTTOM_MID, 0, -20);

/*Cria um novo estilo (azul escuro espesso)*/
static lv_style_t style_line3;
lv_style_copy(&style_line3, &lv_style_plain);
style_line3.line.color = LV_COLOR_MAKE(0x00, 0x3b, 0x75);
style_line3.line.width = 5;

/*Copia a linha anterior e aplica o novo estilo*/
lv_obj_t * line3 = lv_line_create(lv_scr_act(), line1);
lv_line_set_style(line3, &style_line3);
lv_obj_align(line3, line2, LV_ALIGN_OUT_BOTTOM_MID, 0, -20);
```

Mendidor de linha (lv_lmeter)

Escrito para v5.1

Visão geral

O objeto medidor de linha consiste de algumas **linhas radiais** na qual desenha uma escala. Quando se configura um novo valor com `lv_lmeter_set_value(lmeter, new_value)` a parte proporcional da escala será recolorizada.

A função `lv_lmeter_set_range(lmeter, min, max)` atribui o **alcance** do medidor de linha.

Você pode configurar o **ângulo** da escala e o **número de linhas** pelo: `lv_lmeter_set_scale(lmeter, angle, line_num)`. O ângulo padrão é 240 e o padrão do número da linha é 31

Uso do estilo

O medidor de linha usa um estilo na qual pode ser configurado pelo `lv_lmeter_set_style(lmeter, &style)`. As propriedades do medidor de linha são derivados dos seguintes atributos de estilo:

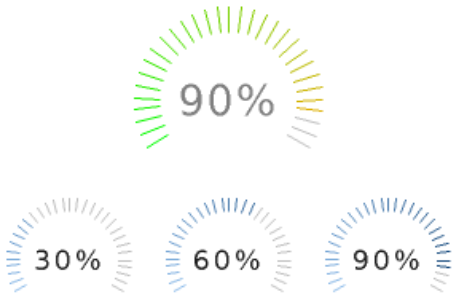
- **line.color** cor da "linha inativa" na qual é maior do que o valor atual
- **body.main_color** cor da "linha ativa" no início da escala
- **body.grad_color** cor da "linha ativa" no fim da escala (gradiente com a cor principal)
- **body.padding.hor** comprimento da linha
- **line.width** comprimento da linha

O estilo padrão é _lv_style_pretty_color_.

Notas

- O medidor de linha não tem background
- É recomendado usar **antialiasing** no menor displays de dpi para mostrar linhas suavizadas
- Números ímpared de escala são melhores
- É melhor se o angulo de escala é: (número de linha - 1) * N, onde N é um inteiro

Exemplo



```
/************************************
 * Cria 3 medidores de linha similares
 ************************************/

/*Cria um estilo simples com comprimento de linha grosso*/
static lv_style_t style_lmeter1;
lv_style_copy(&style_lmeter1, &lv_style_pretty_color);
style_lmeter1.line.width = 2;
style_lmeter1.line.color = LV_COLOR_SILVER;
style_lmeter1.body.main_color = LV_COLOR_HEX(0x91bfed);         /*Azul claro*/
style_lmeter1.body.grad_color = LV_COLOR_HEX(0x04386c);         /*Azul escuro*/

/*Cria o primeiro medidor de linha*/
lv_obj_t * lmeter;
lmeter = lv_lmeter_create(lv_scr_act(), NULL);
lv_lmeter_set_range(lmeter, 0, 100);                    /*Configura o alcance*/
lv_lmeter_set_value(lmeter, 30);                        /*Configura o valor atual*/
lv_lmeter_set_style(lmeter, &style_lmeter1);            /*Aplica o novo estilo*/
lv_obj_set_size(lmeter, 80, 80);
lv_obj_align(lmeter, NULL, LV_ALIGN_IN_BOTTOM_LEFT, 20, -20);

/*Adiciona um rótulo para mostrar o valor atual*/
lv_obj_t * label;
label = lv_label_create(lmeter, NULL);
lv_label_set_text(label, "30%");
lv_label_set_style(label, &lv_style_pretty);
lv_obj_align(label, NULL, LV_ALIGN_CENTER, 0, 0);

/*Cria o segundo medidor de linha e um rótulo*/
lmeter = lv_lmeter_create(lv_scr_act(), lmeter);
lv_lmeter_set_value(lmeter, 60);
lv_obj_align(lmeter, NULL, LV_ALIGN_IN_BOTTOM_MID, 0, -20);

label = lv_label_create(lmeter, label);
lv_label_set_text(label, "60%");
lv_obj_align(label, NULL, LV_ALIGN_CENTER, 0, 0);

/*Cria o terceiro medidor de linha e um rótulo*/
lmeter = lv_lmeter_create(lv_scr_act(), lmeter);
lv_lmeter_set_value(lmeter, 90);
lv_obj_align(lmeter, NULL, LV_ALIGN_IN_BOTTOM_RIGHT, -20, -20);

label = lv_label_create(lmeter, label);
lv_label_set_text(label, "90%");
lv_obj_align(label, NULL, LV_ALIGN_CENTER, 0, 0);

/*********************************
 * Cria um medidor de linha maior
 *********************************/

/*Cria um novo estilo*/
static lv_style_t style_lmeter2;
lv_style_copy(&style_lmeter2, &lv_style_pretty_color);
style_lmeter2.line.width = 2;
style_lmeter2.line.color = LV_COLOR_SILVER;
style_lmeter2.body.padding.hor = 16;            /*Comprimento da linha*/
style_lmeter2.body.main_color = LV_COLOR_LIME;
style_lmeter2.body.grad_color = LV_COLOR_ORANGE;

/*Cria o medidor de linha*/
lmeter = lv_lmeter_create(lv_scr_act(), lmeter);
lv_obj_set_style(lmeter, &style_lmeter2);
lv_obj_set_size(lmeter, 120, 120);
lv_obj_align(lmeter, NULL, LV_ALIGN_IN_TOP_MID, 0, 20);
lv_lmeter_set_scale(lmeter, 240, 31);
lv_lmeter_set_value(lmeter, 90);

/*Cria um estilo de rótulo com uma fonte maior*/
static lv_style_t style_label;
lv_style_copy(&style_label, &lv_style_pretty);
style_label.text.font = &lv_font_dejavu_60;
style_label.text.color = LV_COLOR_GRAY;

/*Adiciona um rótulo para mostrar o valor atual*/
label = lv_label_create(lmeter, label);
lv_label_set_text(label, "90%");
lv_obj_set_style(label, &style_label);
lv_obj_align(label, NULL, LV_ALIGN_CENTER, 0, 0);
```

Rótulo (lv_label)

Escrito para v5.1

Visão geral

Os rótulos são objetos básicos para **mostrar texto**. Não existe limitação no tamanho do texto porque ele é guardado dinamicamente. Você pode modificar o texto em tempo real a qualquer momento com lv_label_set_text().

Você pode usar `\n` para fazer a quebra de linha. Por exemplo: `"line1\nline2\n\nline4"`

O tamaho do objeto rótulo pode ser automaticamente expandido ao tamanho do texto ou o texto pode ser manipulado de acordo com **políticas de modo longo** severas:

- LV_LABEL_LONG_EXPAND: Expande o tamanho do objeto ao tamanho do texto
- LV_LABEL_LONG_BREAK: Mantém o comprimento do objeto, quebra as linhas muito longas e expande a altura do objeto
- LV_LABEL_LONG_DOTS: Mantém o tamanho do objeto, quebra o texto e escreve pontos na última linha
- LV_LABEL_LONG_SCROLL: Expande o tamanho do objeto e desliza o texto no pai (move o objeto rótulo)
- LV_LABEL_LONG_ROLL: Mantém o tamanho e o rolamento somente o texto (não o objeto)

Você pode especificar o modo longo com: `lv_label_set_long_mode(label, long_mode)`

Rótulos são capazes de mostrar texto de um **vetor estático**. Use `lv_label_set_static_text(label, char_array)`. Neste caso, o texto não é guardado na memória dinâmica mas o vetor dado é usado. Mantenha em mente que o vvetor não pode ser uma variável local na qual se destroi quando a função deixa de ser executada.

Você pode também usar um **vetor de caracteres** como um rótulo de texto. O vertor não precisa ser terminado por `\0`. Neste caso, o texto será salvo na memória dinâmica. Para configurar o vetor de caracteres use a função `lv_label_set_array_text(label, char_array)`

O rótulo do **texto pode ser alinhado** à esquerda, direita ou centro com `lv_label_set_align(label, LV_LABEL_ALIGN_LEFT/RIGHT/CENTER)`

Você pode ativar o **desenhar um background** para o rótulo com `lv_label_set_body_draw(label, draw)`

No texto, você pode usar comandos para **recolorizar partes do texto**. `"Write a #ff0000 red# word"`. Por exemplo: Neste recurso pode ser ativado individualmente para cada rótulo pela função `lv_label_set_recolor()`

Os rótulos podem mostrar símbolos ao lado das letras. Aprenda mais sobre símbolos aqui

Os **estilos padrões** dos rótulos é `NULL` assim eles herdam o estilo do pai.

Uso do estilo

- Use todas as propriedades do `style.text`
- Para um desenho background `propriedades style.body` são usadas

Notas

Os **atributos clique ativado do rótulo é desativado** por padrão. Você pode ativar o modo clicar com `lv_obj_set_click(label, true)`

Exemplo


```c
/*Cria rótulo na tela. Por padrão o estilo será herdado do estilo da tela*/
lv_obj_t * title = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(title, "Title Label");
lv_obj_align(title, NULL, LV_ALIGN_IN_TOP_MID, 0, 20);  /*Alinha ao topo*/

/*Cria um novo estilo*/
static lv_style_t style_txt;
lv_style_copy(&style_txt, &lv_style_plain);
style_txt.text.font = &lv_font_dejavu_40;
style_txt.text.letter_space = 2;
style_txt.text.line_space = 1;
style_txt.text.color = LV_COLOR_HEX(0x606060);

/*Cria um novo rótulo*/
lv_obj_t * txt = lv_label_create(lv_scr_act(), NULL);
lv_obj_set_style(txt, &style_txt);                  /*Configura o estilo criado*/
lv_label_set_long_mode(txt, LV_LABEL_LONG_BREAK);     /*Quebra as linhas longas*/
lv_label_set_recolor(txt, true);                    /*Ativa recolorizar por comandos no texto*/
lv_label_set_align(txt, LV_LABEL_ALIGN_CENTER);       /*Centraliza linhas alinhadas*/
lv_label_set_text(txt, "Align lines to the middle\n\n"
                       "#000080 Re-color# #0000ff words of# #6666ff the text#\n\n"
                       "If a line become too long it can be automatically broken into multiple lines");
lv_obj_set_width(txt, 300);                         /*Configuura o comprimento*/
lv_obj_align(txt, NULL, LV_ALIGN_CENTER, 0, 20);      /*Alinha ao centro*/
```

Caixa de mensagem (lv_mbox)

Escrito para v5.1

Visão geral

As caixas de mensagens agem como **pop-ups**. Elas são construídas de um **background**, um **texto** e **botões**. O background é um objeto Contâiner com um ajuste vertical para garantir que o texto e os botões são serão sempre visíveis.

Para **configurar o texto** use a função `lv_mbox_set_text(mbox, "My text")`.

Os botões são uma matriz de Botões. Para **adicionar botões** use a função `lv_mbox_add_btns(mbox, btn_str, action)`. Neste você pode especificar o texto ex.: (`const char * btn_str[] = {"btn1", "btn2", ""}`) e adicionar um callback na qual é chamado quando um botão é pressioando. Para mais informações visite a documentação Matriz de botões.

Com `lv_mbox_start_auto_close(mbox, delay)` a caixa de mensagem pode ser **fechada automaticamente** depois de *delay* milisegundos com uma longa animação. A função `lv_mbox_stop_auto_close(mbox)` irá parar um auto fechamento iniciado.

O tempo da animação fechar pode ser ajustado pelo `lv_mbox_set_anim_time(mbox, anim_time)`.

Uso do estilo

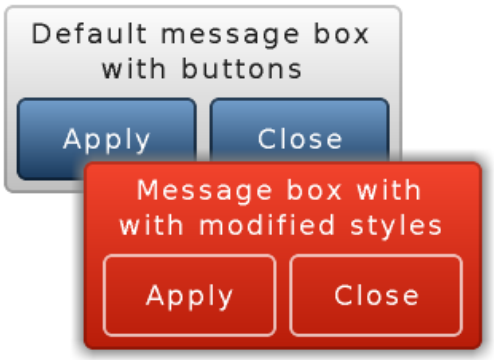
Use `lv_mbox_set_style(mbox, LV_MBOX_STYLE_..., &style)` para configurar um novo estilo para um elemento da caixa de mensagem:

- **LV_MBOX_STYLE_BG** especifica o background do estilo do contâiner. *style.body* para o backgrounde *style.label* para a aparência do texto. Padrão: lv_style_pretty
- **LV_MBOX_STYLE_BTN_BG** estilo dos botões background da (matriz de botões). Padrão: lv_style_transp
- **LV_MBOX_STYLE_BTN_REL** estilo dos botões pressionados. Padrão: lv_style_btn_rel
- **LV_MBOX_STYLE_BTN_PR** estilo dos botões pressionados. Padrão: lv_style_btn_pr
- **LV_MBOX_STYLE_BTN_TGL_REL** estilo dos botões comutados deixados de pressioanar. Padrão: lv_style_btn_tgl_rel
- **LV_MBOX_STYLE_BTN_TGL_PR** estilo dos botões pressionados. Padrão: lv_style_btn_tgl_pr
- **LV_MBOX_STYLE_BTN_INA** estilo dos botões inativos. Padrão: lv_style_btn_ina

Notas

- A **altura dos botões** vem do *font height* + 2 x *body.vpad* de _LV_MBOX_STYLE_BTN_REL_

Exemplo



```c
/*Chamado quando um botão é clicado*/
static lv_res_t mbox_apply_action(lv_obj_t * mbox, const char * txt)
{
    printf("Mbox button: %s\n", txt);

    return LV_RES_OK; /*Retorna OK se a mensagem não é apagada*/
}

.
.
.
.

/*********************************
 * Cria uma caixa de mensagem padrão
 *********************************/

lv_obj_t * mbox1 = lv_mbox_create(lv_scr_act(), NULL);
lv_mbox_set_text(mbox1, "Default message box\n"
                        "with buttons");                    /*Configura o texto*/
/*Add two buttons*/
static const char * btns[] ={"\221Apply", "\221Close", ""}; /*Descrição do botão. '\221' controle de caractere lv_btnm*/
lv_mbox_add_btns(mbox1, btns, NULL);
lv_obj_set_width(mbox1, 250);
lv_obj_align(mbox1, NULL, LV_ALIGN_IN_TOP_LEFT, 10, 10); /*Alinha ao canto*/

/*********************************************
 * Cria uma caixa de mensagem com novos estilos
 *********************************************/

/*Cria um novo estilo de background*/
static lv_style_t style_bg;
lv_style_copy(&style_bg, &lv_style_pretty);
style_bg.body.main_color = LV_COLOR_MAKE(0xf5, 0x45, 0x2e);
style_bg.body.grad_color = LV_COLOR_MAKE(0xb9, 0x1d, 0x09);
style_bg.body.border.color = LV_COLOR_MAKE(0x3f, 0x0a, 0x03);
style_bg.text.color = LV_COLOR_WHITE;
style_bg.body.padding.hor = 12;
style_bg.body.padding.ver = 8;
style_bg.body.shadow.width = 8;

/*Cria um estilo de botões pressionado e deixado de pressionar*/
static lv_style_t style_btn_rel;
static lv_style_t style_btn_pr;
lv_style_copy(&style_btn_rel, &lv_style_btn_rel);
style_btn_rel.body.empty = 1;                       /*Desenha somente a borda*/
style_btn_rel.body.border.color = LV_COLOR_WHITE;
style_btn_rel.body.border.width = 2;
style_btn_rel.body.border.opa = LV_OPA_70;
style_btn_rel.body.padding.hor = 12;
style_btn_rel.body.padding.ver = 8;

lv_style_copy(&style_btn_pr, &style_btn_rel);
style_btn_pr.body.empty = 0;
style_btn_pr.body.main_color = LV_COLOR_MAKE(0x5d, 0x0f, 0x04);
style_btn_pr.body.grad_color = LV_COLOR_MAKE(0x5d, 0x0f, 0x04);

/*Copia a caixa de mensagem (Os botões serão copiados também)*/
lv_obj_t * mbox2 = lv_mbox_create(lv_scr_act(), mbox1);
lv_mbox_set_text(mbox2, "Message box with\n"
                        "with modified styles");
lv_mbox_set_style(mbox2, LV_MBOX_STYLE_BG, &style_bg);
lv_mbox_set_style(mbox2, LV_MBOX_STYLE_BTN_REL, &style_btn_rel);
lv_mbox_set_style(mbox2, LV_MBOX_STYLE_BTN_PR, &style_btn_pr);
lv_obj_align(mbox2, mbox1, LV_ALIGN_OUT_BOTTOM_LEFT, 50, -20);   /*Alinha de acordo com a caixa de mensagem prévia*/
```

Página (lv_page)

Escrito para v5.1

Visão geral

A Página consiste de dois contâiner em cada outro: o botão é o **background** (ou base) e o top é o **deslizável**. Se você criou um filho na página ele será automaticamente movido em um contâiner deslizável. Se o contâiner deslizável se tornar maior do que o background ele **pode ser deslizável pelo arrasto** (como a lista de smart phones).

Por padrão o atributo *auto fit* do atributo deslizável é ativado verticalmente assim sua altura será aumentada para incluir seus filhos. O comprimento do deslizável é automaticamente ajustado pelo comprimento do background (menos o espaçamento horizontal do background).

O objeto background pode ser referenciado como a página dele mesmo como: `lv_obj_set_width(page, 100)`.

O objeto de rolagem pode ser recuperada com: `lv_page_get_scrl(page)`.

Barra de rolagem pode ser mostrado de acordo a quatro políticas:

- LV_SB_MODE_OFF: Nunca mostra barra de rolagem
- LV_SB_MODE_ON: Sempre mostra barra de rolagem
- LV_SB_MODE_DRAG: Mostra barra de rolagens quando a pagina está sendo arrastada
- LV_SB_MODE_AUTO: Mostra barra de rolagem quando o contâiner deslizável é longo o bastante para ser deslizado

Você pode configurar a barra de rolagem mostranto política pelo: `lv_page_set_sb_mode(page, SB_MODE)`. O valor padrão é _LV_PAGE_SB_MODE_ON_;

Você pode **colar um filho** para a página. Neste caso você pode deslizar a página com arrasto do objeto filho. Ele pode ser ativado pelo `lv_page_glue_obj(child, true)`.

Você pode **focar um objeto** sobre uma página com: `lv_page_focus(page, child, anim_time)`.

Ele moverá o contâiner deslizável para mostrar um filho. Se o último parâmetro não é zero então a página se moverá com uma animação.

Uma ação ao soltar e ao pressionar podem ser assinadas para a Página com `lv_page_set_rel_action(page, my_rel_action)` e `lv_page_set_pr_action(page, my_pr_action)`. A ação pode ser gatihada de um Background e um objeto deslizável também.

Existem funções para direcionar **atribuir/obter atributos dos deslizáveis**:

- `lv_page_set_scrl_fit()`
- `lv_page_set_scrl_width()`
- `lv_page_set_scrl_height()`
- `lv_page_set_scrl_layout()`

Uso do estilo

Use `lv_page_set_style(page, LV_PAGE_STYLE_..., &style)` para atribuir um novo estilo para um elemento da página:

- **LV_PAGE_STYLE_BG** estilo do background na qual usa todas as propriedades *style.body* (padrão: lv_style_pretty_color)
- **LV_PAGE_STYLE_SCRL** estilo deslizável na qual usa todas as propriedades *style.body* (padrão: lv_style_pretty)
- **LV_PAGE_STYLE_SB** estilo da barra deslizável na qual usa todas as propriedades *style.body*. Espaçamentos horizontal/vertical configura o espaçamento da barra de rolagem respectivamente a o espaçamento interno configura o comprimento da barra de rolagem (padrão: lv_style_pretty_color)

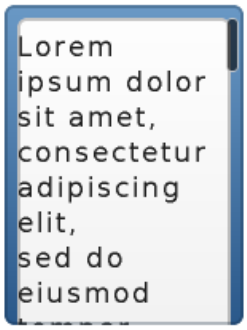
Notas

- **Atribuindo a posição do fillho** não é possível em direções x ou y se o ajuste correspondente *hor* ou *ver* estão ativados. É porque se a coordenada _y_ está modificada (com *ajuste ver* ativado) o objeto deslizável será redimensionado para estar diratamente acima e abaixo do

filho. Mas uma parte deslizável não pode estar no meio assim ele será movido de volta ao topo. Para evitar isso use `lv_obj_align()` para trocar o objeto relatiovo a cada outro (um tem que estar no topo/esquerda) ou desativado o ajuste com `lv_page_set_scrl_fit(page, false, false);` e configure seu tamanho `lv_page_set_scrl_width/height(page, 100)`.

- O background desenha sua borda quando o deslizável é desenhado. Ele garante que a página sempre terá que ser uma forma fechada até mesmo se um deslizável tem a mesma cor da página pai

Exemplo




```
/*Cria um estilo de barra*/
static lv_style_t style_sb;
lv_style_copy(&style_sb, &lv_style_plain);
style_sb.body.main_color = LV_COLOR_BLACK;
style_sb.body.grad_color = LV_COLOR_BLACK;
style_sb.body.border.color = LV_COLOR_WHITE;
style_sb.body.border.width = 1;
style_sb.body.border.opa = LV_OPA_70;
style_sb.body.radius = LV_RADIUS_CIRCLE;
style_sb.body.opa = LV_OPA_60;
style_sb.body.padding.hor = 3;          /*Espaçamento horizontal à direita*/
style_sb.body.padding.inner = 8;        /*Comprimento da barra de rolagem*/

/*Cria uma página*/
lv_obj_t * page = lv_page_create(lv_scr_act(), NULL);
lv_obj_set_size(page, 150, 200);
lv_obj_align(page, NULL, LV_ALIGN_CENTER, 0, 0);
lv_page_set_style(page, LV_PAGE_STYLE_SB, &style_sb);           /*Configura o estilo da barra de rolagem*/
lv_page_set_sb_mode(page, LV_SB_MODE_AUTO);                     /*Mostra que o deslizamento da barra de rolagem é possível*/

/*Cria um rótulo na página*/
lv_obj_t * label = lv_label_create(page, NULL);
lv_label_set_long_mode(label, LV_LABEL_LONG_BREAK);             /*Automaticamente quebra longas linhas*/
lv_obj_set_width(label, lv_page_get_scrl_width(page));          /*Configura o comprimento. Linhas serão quebradas aqui*/
lv_label_set_text(label, "Lorem ipsum dolor sit amet, consectetur adipiscing elit,\n"
                         "sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.\n"
                         "Ut enim ad minim veniam, quis nostrud exercitation ullamco\n"
                         "laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure\n"
                         "dolor in reprehenderit in voluptate velit esse cillum dolore\n"
                         "eu fugiat nulla pariatur.\n"
                         "Excepteur sint occaecat cupidatat non proident, sunt in culpa\n"
                         "qui officia deserunt mollit anim id est laborum.");
```

Pré-carregador (lv_preload)

Escrito para v5.2

Visão geral

O objeto pré-carregador é **um arco girante sobre uma borda**.

O **comprimento do arco** pode ser ajustado pelo `lv_preload_set_arc_length(preload, deg)`.

A **velocidade do giro** pode ser ajustada pelo `lv_preload_set_spin_time(preload, time_ms)`.

Uso do estilo

O estilo `LV_PRELOAD_STYLE_MAIN` descreve ambos estilos de arco e a borda:

- **arco** é descrito pela propriedade de linha
- **borda** é descrito pela propriedade `body.border` incluindo `body.padding.hor/ver` (o menor é usado) para dar um menor raio para a borda.

Exemplo

```
/*Cria um estilo para o Pré-carregador*/
static lv_style_t style;
lv_style_copy(&style, &lv_style_plain);
style.line.width = 10;                          /*arco de espessura 10 px*/
style.line.color = LV_COLOR_HEX3(0x258);        /*Cor do arco azul*/

style.body.border.color = LV_COLOR_HEX3(0xBBB); /*Cor do arco cinza*/
style.body.border.width = 10;
style.body.padding.hor = 0;

/*Cria um objeto pré-carregador*/
lv_obj_t * preload = lv_preload_create(lv_scr_act(), NULL);
lv_obj_set_size(preload, 100, 100);
lv_obj_align(preload, NULL, LV_ALIGN_CENTER, 0, 0);
lv_preload_set_style(preload, LV_PRELOAD_STYLE_MAIN, &style);
```

Rolagem (lv_roller)

Escrito para v5.1 Atualizado para v5.2

Visão geral

A rolagem permite a você simplesmente **selecionar uma opção de várias** com deslizamento. Suas funcionalidades são similares ao Lista suspensa.

As **opções** são passadas para a Rolagem como uma **string** com `lv_roller_set_options(roller, options)`. As opções devem ser separadas pelo `\n`. Por exemplo: `"First\nSecond\nThird"`.

Você pode **selecionar uma opção manualmente** com `lv_roller_set_selected(roller, id)`, onde _id_ é o índice de uma opção.

Uma **função callback** pode ser especificada com `lv_roller_set_action(roller, my_action)` para chamar quando uma nova opção é selecionada.

A **altura** da rolagem pode ser ajustada com `lv_roller_set_visible_row_count(roller, row_cnt)` para configurar o número de opções visíveis.

O **comprimento** é ajustado automaticamente. Para evitar isso, aplique `lv_roller_set_hor_fit(roller, false)` e configure o comprimento manualmente pelo `lv_obj_set_width(roller, width)`. Você deve usar `lv_roller_set_hor_fit(roller, false)` ao invés de `lv_cont_set_fit(lv_page_get_scrl(roller), false, false);`, por outro lado você obterá um estilo LV_LABEL_ALIGN_LEFT do texto da lista do rótulo.

O tempo da **animação** abrir/fechar da Lista suspensa é ajusdado pelo `lv_roller_set_anim_time(roller, anim_time)`. Animação zero siginifica nenhuma animação. Esta característica é implementada junto com lv_ddlist.c na v5.2: `lv_ddlist_set_anim_time(roller, anim_time);` deve ser usado para animação.

Uso do estilo

O `lv_roller_set_style(roller, LV_ROLLER_STYLE_..., &style)` configura o estilo de uma rolagem.

- **LV_ROLLER_STYLE_BG** Estilo do background. Todos as propriedades do *style.body* são usadas. Ele é usado para o estilo do rótulo de *style.text*. Gradiente é aplicado no topo e no fundo também. Padrão: _lv_style_pretty_
- **LV_DDLIST_STYLE_SEL** Estilo da opção selecionada. As propriedades *style.body* são usadas. A opção selecionada será recolorida com *text.color*. Padrão: _lv_style_plain_color_

Exemplo

```
/*Cria uma lista suspensa padrão*/
lv_obj_t *roller1 = lv_roller_create(lv_scr_act(), NULL);
lv_roller_set_options(roller1, "Apple\n"
                               "Broccoli\n"
                               "Cabbage\n"
                               "Dewberry\n"
                               "Eggplant\n"
                               "Fig\n"
                               "Grapefruit");
lv_obj_set_pos(roller1, 50, 80);


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

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

/*Cria uma lista suspensa e aplica novos estilos*/
lv_obj_t *roller2 = lv_roller_create(lv_scr_act(), NULL);
lv_roller_set_options(roller2, "0\n"
                               "1\n"
                               "2\n"
                               "3\n"
                               "4\n"
                               "5\n"
                               "6\n"
                               "7\n"
                               "8\n"
                               "9");
lv_roller_set_style(roller2, LV_ROLLER_STYLE_BG, &bg_style);
lv_roller_set_selected(roller2, 3, false);
lv_roller_set_style(roller2, LV_ROLLER_STYLE_SEL, &sel_style);
lv_roller_set_visible_row_count(roller2, 3);
lv_roller_set_hor_fit(roller2, false);
lv_obj_set_width(roller2, 40);
lv_obj_set_pos(roller2, 220, 50);
```

Controle deslizante (lv_slider)

Escrito para v5.1

Visão geral

O objeto Controle Deslizante parece como uma **Barra** suplementada **com uma Alça**. A Alça pode ser **arrastada para configurar um valor**. O controle deslizante pode ser vertical ou horizontal.

Para configurar um **valor inicial** use a função `lv_slider_set_value(slider, new_value)` ou `lv_slider_set_value_anim(slider, new_value, anim_time)` para configurar o valor com uma animação.

Para especificar o **alcance** (valores min, máx) o `lv_slider_set_range(slider, min , max)` pode ser usado.

Uma **função callback** pode ser assinada para chamar quando um novo valor é configurado pelo usuário `lv_slider_set_action(slider, my_action)`.

A **alça pode ser trocada** de dois jeitos:

- dentro do background sobre valores min/máx
- sobre as bordas sobre valores mim/máx

Use o `lv_slider_set_knob_in(slider, true/false)` para escolher entre os modos. (*knob_in == false* é o padrão)

Uso do estilo

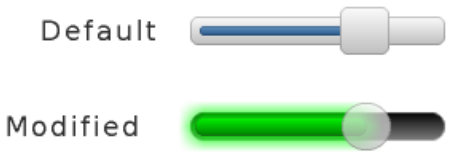
Você pode modificar o estilo do controle deslizante com `lv_slider_set_style(slider, LV_SLIDER_STYLE_..., &style)`.

- **LV_SLIDER_STYLE_BG** Estilo do background. Todas as propriedades *style.body* são usadas. O valor *padding* faz o controle deslizante menor do que a alça (valor negativo faz dele maior)
- **LV_SLIDER_STYLE_INDIC** Estilo do indicador. Todas as propriedades *style.body* são usadas. O valores *padding* fazem o indicador menor do que o background.
- **LV_SLIDER_STYLE_KNOB** Estilo da alça. As propriedades *style.body* são usadas exceto espaçamento

Notas

- A alça não é um objeto real, ele é somente desenhado acima da Barra

Exemplo



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

    return LV_RES_OK;
}

.
.
.

/*Cria um controle deslizante padrão*/
lv_obj_t * slider1 = lv_slider_create(lv_scr_act(), NULL);
lv_obj_set_size(slider1, 160, 30);
lv_obj_align(slider1, NULL, LV_ALIGN_IN_TOP_RIGHT, -30, 30);
lv_slider_set_action(slider1, slider_action);
lv_bar_set_value(slider1, 70);


/*Criar um rótulo à direita ao controle deslizante*/
lv_obj_t * slider1_label = lv_label_create(lv_scr_act(), NULL);
lv_label_set_text(slider1_label, "Default");
lv_obj_align(slider1_label, slider1, LV_ALIGN_OUT_LEFT_MID, -20, 0);

/*Criai uma barra, um indicador e um estilo de alça*/
static lv_style_t style_bg;
static lv_style_t style_indic;
static lv_style_t style_knob;

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

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

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

/*Cria um segundo controle deslizante*/
lv_obj_t * slider2 = lv_slider_create(lv_scr_act(), slider1);
lv_slider_set_style(slider2, LV_SLIDER_STYLE_BG, &style_bg);
lv_slider_set_style(slider2, LV_SLIDER_STYLE_INDIC,&style_indic);
lv_slider_set_style(slider2, LV_SLIDER_STYLE_KNOB, &style_knob);
lv_obj_align(slider2, slider1, LV_ALIGN_OUT_BOTTOM_MID, 0, 30); /*Alinha abaixo da 'bar1'*/

/*Cria um segundo rótulo*/
lv_obj_t * slider2_label = lv_label_create(lv_scr_act(), slider1_label);
lv_label_set_text(slider2_label, "Modified");
lv_obj_align(slider2_label, slider2, LV_ALIGN_OUT_LEFT_MID, -30, 0);
```

Caixa girante (lv_spinbox)

Escrito para v5.3

- Este é um trabalho em progresso

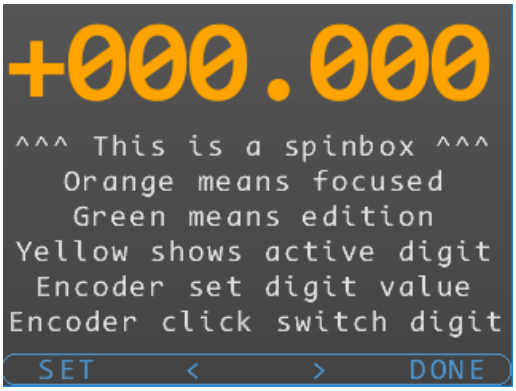
Visão geral

Uso do estilo

Notas

- nota

Exemplo




```
static void spinbox_value_changed(lv_obj_t * spinbox)
{

}

.
.
.
spinbox = lv_spinbox_create(lv_scr_act(), NULL);
lv_spinbox_set_style(spinbox, LV_SPINBOX_STYLE_BG, &spinBoxStyle);
lv_spinbox_set_style(spinbox, LV_SPINBOX_STYLE_CURSOR, &spinBoxCursorStyle);
lv_obj_set_size(spinbox, LV_HOR_RES, 80);
lv_obj_align(spinbox, NULL, LV_ALIGN_IN_TOP_LEFT, 4, 0);
```

Interruptor (lv_sw)

Escrito para v5.1, revisão 2

Visão geral

A chave pode ser usado para **ligar/desligar** algo. A aparência se assemelha a um deslizador. O estado da chave pode ser mudado quando:

- Clicando sobre ele
- Deslizando ele
- Usando as funções `lv_sw_on(sw)` e `lv_sw_off(sw)`

Uma **função callback** pode ser assinada para chamar quando o usuário usa a chave: `lv_sw_set_action(sw, my_action)`.

Novo em v5.3: Chaves podem ser animadas ao ser chamada por `lv_sw_set_anim_time(sw, anim_ms)`.

Uso do estilo

Você pode modificar o estilo da chave com `lv_sw_set_style(sw, LV_SW_STYLE_..., &style)`.

- **LV_SW_STYLE_BG** Estilo do background. Todas as propriedades de *style.body* são usadas. O valores de *padding* fazem a Chave menor do que a alça. (valor negativo faz ela mais larga)
- **LV_SW_STYLE_INDIC** Estilo do indicador. Todas as propriedades *style.body* são usadas. Os valores de *padding* fazem o indicador menor do que o background.
- **LV_SW_STYLE_KNOB_OFF** Estilo da alça quando ela estiver desligada. A propriedade *style.body* é usada exceto o espaçamento.
- **LV_SW_STYLE_KNOB_ON** Estilo da alça quando a chave está ativada. A propriedade *style.body* é usada exceto o espaçamento.

Notes

- A alça não é um objeto real ele somente é um desenho acima da barra

Exemplo

```
/*Cria estilo para a chave*/
static lv_style_t bg_style;
static lv_style_t indic_style;
static lv_style_t knob_on_style;
static lv_style_t knob_off_style;
lv_style_copy(&bg_style, &lv_style_pretty);
bg_style.body.radius = LV_RADIUS_CIRCLE;

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

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

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

/*Cria uma chave e aplica os estilos*/
lv_obj_t *sw1 = lv_sw_create(lv_scr_act(), NULL);
lv_sw_set_style(sw1, LV_SW_STYLE_BG, &bg_style);
lv_sw_set_style(sw1, LV_SW_STYLE_INDIC, &indic_style);
lv_sw_set_style(sw1, LV_SW_STYLE_KNOB_ON, &knob_on_style);
lv_sw_set_style(sw1, LV_SW_STYLE_KNOB_OFF, &knob_off_style);
lv_obj_align(sw1, NULL, LV_ALIGN_CENTER, 0, -50);

/*Copia a primeira chave e o liga*/
lv_obj_t *sw2 = lv_sw_create(lv_scr_act(), sw1);
lv_sw_set_on(sw2);
lv_obj_align(sw2, NULL, LV_ALIGN_CENTER, 0, 50);
```

Visualizador de tab (lv_tabview)

Escrito para v5.1

Visão geral

O objeto Visualizador de Tab pode ser usado para **organizar conteúdos em tabulações**. Você pode **adicionar uma nova tab** com `lv_tabview_add_tab(tabview, "Tab name")`. Ele retornará com um ponteiro para um objeto Página onde você pode adicionar o conteúdo das tabulações.

Para **selecionar um tab** você pode:

- Clicar nele na parte do cabeçalho
- Deslizar horizontalmente
- Use a função `lv_tabview_set_tab_act(tabview, id, anim_en)`

O **deslizamento manual** pode ser desativado com `lv_tabview_set_sliding(tabview, false)`.

O **tempo de animação** é ajustado pelo `lv_tabview_set_anim_time(tabview, anim_time)`.

Uma **função callback** pode ser assinado para evento **carga de tab** com `lv_tabview_set_tab_load_action(tabview, action)`. A função de chamada precisa ter os seguintes protótipos:

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

Onde _act_id_ significa tabulação na qual será carregada. Na ação `lv_tabview_get_tab_act(tabview)` dará a id da tabulação antiga.

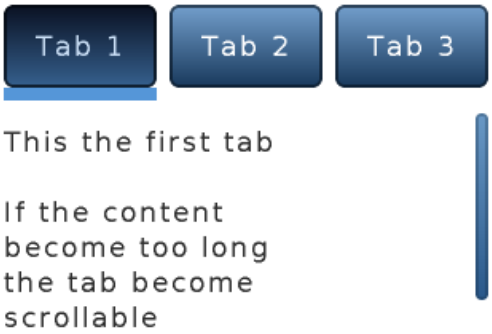
Uso do estilo

Use `lv_tabview_set_style(tabview, LV_TABVIEW_STYLE_..., &style)` para configurar um novo estilo para um elemento do novo visualizador de tabulação:

- **LV_TABVIEW_STYLE_BG** principal background na qual usa todas as propriedades *style.body* (padrão: lv_style_plain)
- **LV_TABVIEW_STYLE_INDIC** um fino retângulo no topo para indicar a tabulação atual. Use todas as propriedades *style.body*. Sua altura vem de *body.padding.inner* (padrão: _lv_style_plain_color_)
- **LV_TABVIEW_STYLE_BTN_BG** estilo dos botões de tabulação do background. Use todas as propriedades do *style.body*. A altura do cabeçalho será configurado automaticamente considerando *body.padding.ver* (padrão: _lv_style_transp_)
- **LV_TABVIEW_STYLE_BTN_REL** estilo dos botões de tabulação soltos. Use todas propriedades *style,body*. (padrão: _lv_style_tbn_rel_)
- **LV_TABVIEW_STYLE_BTN_PR** estilo dos botões de tabulação pressionado. Use todas propriedades *style.body*. (padrão _lv_style_tbn_rel_)
- **LV_TABVIEW_STYLE_BTN_TGL_REL** estilo de botões de tabulação comutado. Use todas propriedades *style.body*. (padrão: _lv_style_tbn_rel_)
- **LV_TABVIEW_STYLE_BTN_TGL_PR** estilo de botões comutados pressionados. Use todas as propriedades *style.body*. (padrão: _lv_style_btn_tgl_pr_)

Notas

Exemplo




```c
/*Cria um objeto visualizador de tabulação*/
lv_obj_t *tabview;
tabview = lv_tabview_create(lv_scr_act(), NULL);

/*Adiciona 3 tabulações (as tabulações são páginas (lv_page) e pode ser deslizado*/
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");


/*Adiciona conteúdo para as tabulações*/
lv_obj_t * label = lv_label_create(tab1, NULL);
lv_label_set_text(label, "This the first tab\n\n"
                         "If the content\n"
                         "become too long\n"
                         "the tab become\n"
                         "scrollable\n\n");

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

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

Área de texto (lv_ta)

Escrito para v5.1

Visão geral

A Área de Texto é **uma página** com um **rótulo** e um **cursor** nele. Você pode **inserir texto ou caracteres** para a posição atual com:

- `lv_ta_add_char(ta, 'c')`
- `lv_ta_add_text(ta, "insert this text")`

O `lv_ta_set_text(ta, "New text")` **muda o texto todo**.

Para **apagar um caractere** da esquerda da posição do cursor use `lv_ta_del_char(ta)`.

A posição do cursor pode ser modificada diretamente como `lv_ta_set_cursor_pos(ta, 10)` ou por passo:

- `lv_ta_cursor_right(ta)`
- `lv_ta_cursor_left(ta)`
- `lv_ta_cursor_up(ta)`
- `lv_ta_cursor_down(ta)`

Existem vários tipos de cursores. Você pode configurar um deles com: `lv_ta_set_cursor_type(ta, LV_CURSOR_...)`

- LV_CURSOR_NONE
- LV_CURSOR_LINE
- LV_CURSOR_BLOCK
- LV_CURSOR_OUTLINE
- LV_CURSOR_UNDERLINE

Você pode 'OR' _LV_CURSOR_HIDDEN_ para qualquer tipo para esconder o cursor.

A área de texto pode ser configurada para ser uma linha com `lv_ta_set_one_line(ta, true)`.

A área de texto suporta **modo de senha**. Ele pode ser ativado com `lv_ta_set_pwd_mode(ta, true)`.

Uso do estilo

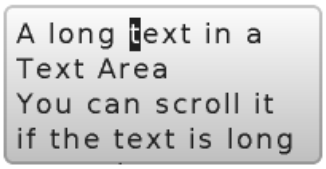
Use `lv_ta_set_style(page, LV_TA_STYLE_..., &style)` para configurar um novo estilo para um elemento da área de texto:

- **LV_TA_STYLE_BG** estilo do background na qual usa todas as propriedades *style.body*. O rótulo também usa esse *style.label* deste estilo. (padrão: lv_style_pretty)
- **LV_TA_STYLE_SB** estilo de barra de rolagem na qual usa todas as propriedades do *style.body* (padrão: lv_style_transp)
- **LV_TA_STYLE_CURSOR** estilo de cursor. Se NULL então a biblioteca configura para nós um estilo automaticamente de acordo com a cor do rótulo e fonte
  - LV_CURSOR_LINE: uma longa *style.line.width* linha mais desenhado como um retangulo como espaçamento *style.body* horizontais e verticais dá um espaço sobre o cursor
  - LV_CURSOR_BLOCK: um retangulo como espaçamentos *style.body* horizontais e verticais que fazem o retangulo mais largo
  - LV_CURSOR_OUTLINE: um retangulo vazio (somente uma borda) como espaçamentos *style.body* horizontais e verticais que fazem o retangulo mais largo
  - LV_CURSOR_UNDERLINE: uma linha de comprimento *style.line.width* mas desenhada como retangulo como espaçamentos horizontais e verticais que fazem um offset sobre o cursor

Notas

- No modo senha `lv_ta_get_text(ta)` te dá o texto real e não o caractere asteristico

Exemplo




```
/*Cria um estilo de barra de rolagem*/
static lv_style_t style_sb;
lv_style_copy(&style_sb, &lv_style_plain);
style_sb.body.main_color = LV_COLOR_BLACK;
style_sb.body.grad_color = LV_COLOR_BLACK;
style_sb.body.border.color = LV_COLOR_WHITE;
style_sb.body.border.width = 1;
style_sb.body.border.opa = LV_OPA_70;
style_sb.body.radius = LV_RADIUS_CIRCLE;
style_sb.body.opa = LV_OPA_60;

/*Cria uma área de texto normal*/
lv_obj_t * ta1 = lv_ta_create(lv_scr_act(), NULL);
lv_obj_set_size(ta1, 200, 100);
lv_obj_align(ta1, NULL, LV_ALIGN_CENTER, 0, - LV_DPI / 2);
lv_ta_set_style(ta1,LV_TA_STYLE_SB, &style_sb);                     /*Aplica o estilo da barra de rolagem*/
lv_ta_set_cursor_type(ta1, LV_CURSOR_BLOCK);
lv_ta_set_text(ta1, "A text in a Text Area\n"
               "You can scroll it if the text is long enough.");    /*Atribui um texto inicial*/
lv_ta_set_cursor_pos(ta1, 2);                                       /*Configura a posição do cursor*/
lv_ta_add_text(ta1, "long ");                                       /*Insere uma palavra a posição atual do cursor*/

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

/*Cria um teste alinhado no modo senha*/
lv_obj_t * ta2 = lv_ta_create(lv_scr_act(), ta1);
lv_obj_align(ta2, ta1, LV_ALIGN_OUT_BOTTOM_MID, 0, 50);
lv_ta_set_style(ta2,LV_TA_STYLE_BG, &style_bg);                     /*Aplica o estilo do background*/
lv_ta_set_one_line(ta2, true);
lv_ta_set_cursor_type(ta2, LV_CURSOR_LINE);
lv_ta_set_pwd_mode(ta2, true);
lv_ta_set_text(ta2, "Password");
```

Janela (lv_window)

Escrito para v5.1

Visão geral

As janelas são um dos **mais complexos** contâiner similares a objetos. Eles são construídos de duas partes principais: um **cabeçalho** Contâiner no topo e uma Página para o **conteúdo** abaixo do cabeçalho.

Sobre o cabeçalho há um **título** na qual pode ser modificado por: `lv_win_set_title(win, "New title")`. O título sempre herda o estilo do cabeçalho.

Você pode adicionar **botões de controle** para o lado direito do cabeçalho com: `lv_win_add_btn(win, "U:/close", my_close_action)`. O segundo parâmetro é um atalho de arquivo de imagem, o terceiro parâmetro é uma função para chamar quando o botão é solto. Você pode usar **símbolos** como imagens também como: `lv_win_add_btn(win, SYMBOL_CLOSE, my_close_action)`.

Você pode modificar o **tamanho do controle de botões** com a função `lv_win_set_btn_size(win, new_size)`.

O comportamento da barra de rolagem pode ser configurado pelo `lv_win_set_sb_mode(win, LV_SB_MODE_...)`.

Para confugurar um layout para o conteúdo use `lv_win_set_layout(win, LV_LAYOUT_...)`.

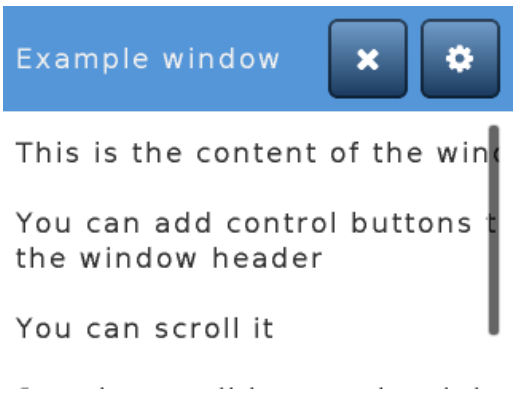
Uso do estilo

Use `lv_win_set_style(win, LV_WIN_STYLE_..., &style)` para configurar um novo estilo para um elemento da janela:

- **LV_WIN_STYE_BG** background principal na qual usa todas as propriedades de *style.body* (cabeçalho e página de conteúdo são colocadas nele) (padrão: lv_style_plain)
- **LV_WIN_STYLE_CONTENT_BG** conteúdo do background da página na qual usa todas as propriedades de *style.body* (padrão: lv_style_transp)
- **LV_WIN_STYLE_CONTENT_SCRL** conteúdo da página da parte deslizável na qual usa todas as propriedades de *style.body* (padrão lv_style_transp)
- **LV_WIN_STYLE_SB** estilo da barra de deslizamento na qual usa todas as propriedades de *style.body*. Espaçamentos hor/ver* configuram o espaçamento da barra de rolagem respectivamente e seus espaçamentos internos configuram o comprimento da barra de rolagem. (padrão: lv_style_pretty_color)
- **LV_WIN_STYLE_HEADER** estilo do cabeçalho na qual usa todas as propriedades de *style.body* (padrão: lv_style_plain_color)
- **LV_WIN_STYLE_BTN_REL** estilo do botão solto (sobre o cabeçalho) na qual usa todas as propriedades de *style.body* (padrão: lv_style_btn_rel)
- **LV_WIN_STYLE_BTN_PR** estilo do botão solto (sobre o cabeçalho) na qual usa todas as propriedades de *style.body* (padrão: lv_style_btn_pr)

Notas

Exemplo



```c
/*Cria um estilo de uma barra de rolagem*/
static lv_style_t style_sb;
lv_style_copy(&style_sb, &lv_style_plain);
style_sb.body.main_color = LV_COLOR_BLACK;
style_sb.body.grad_color = LV_COLOR_BLACK;
style_sb.body.border.color = LV_COLOR_WHITE;
style_sb.body.border.width = 1;
style_sb.body.border.opa = LV_OPA_70;
style_sb.body.radius = LV_RADIUS_CIRCLE;
style_sb.body.opa = LV_OPA_60;

/*Cria uma janela*/
lv_obj_t * win = lv_win_create(lv_scr_act(), NULL);
lv_win_set_title(win, "Example window");                        /*Configura o título*/
lv_win_set_style(win, LV_WIN_STYLE_SB, &style_sb);              /*Configura o estilo da barra de rolagem*/


/*Adiciona botão de controle ao cabeçalho*/
lv_win_add_btn(win, SYMBOL_SETTINGS, my_setup_action);            /*Adiciona um botão de configuração*/
lv_win_add_btn(win, SYMBOL_CLOSE, lv_win_close_action);           /*Adiciona botão fechar e usa a ação fechar embutida*/


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