libGUI Documentation

Release 0.8.0

ANSSI

May 17, 2019

CONTENTS

Contents

This library is an easy to use, tile-based graphical interface manager. It aim to support small screens with potential low refresh speed.

It permit to define tiles and menus, with easy to configure API for embedded systems.

The current implementation of libGUI is compatible with the ili9341 TFT screen driver and ad7843 touchscreen driver of the Wookey project, but can be easily ported to other tft/touchscreen drivers, while the required primitives are defined in the corresponding drivers.

CHAPTER

ONE

OVERVIEW

The graphical user interface is based on a succession of menus and tiles declaration.

- Each menu is independent and can be accessed from a tile touch.
- Each tile is associated to a given menu and has various properties
  - A width
  - A height
  - A background and a foreground color
  - A text content
  - An icon
  - An associated action

Each GUI element (menu or tile) is associated to a *descriptor*, which is returned at the element declaration. This descriptor is used during the lifetime of the element to update it (e.g. adding a tile to a given menu descriptor, or updating a tile content).

The tile positioning in a menu is automatically done, from top to bottom of the screen, from left to right, depending the each successive declared tile size of a given menu.

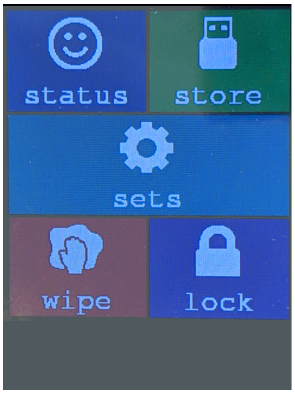
For example, if you declare, for a given menu:

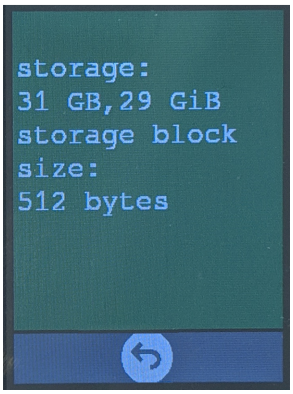
- A tile of size (TILE_WIDTH_FULL, TILE_HEIGHT_STD)
- A tile of size (TILE_WIDTH_HALF, TILE_HEIGHT_STD)
- A tile of size (TILE_WIDTH_HALF, TILE_HEIGHT_STD)
- A Tile of size (TILE_WIDTH_FULL, TILE_HEIGHT_DOUBLE)

The resulting menu would look like:

The libGUI tiles and menus API rendering looks like (effective interface of Wookey GUI):

Caution: By now, reduced sized tiles (for e.g. WIDTH_HALF or WIDTH_THIRD) are supported only for TILE_HEIGHT_STD height)


status
store
sets
wipe
lock


storage:
31 GB,29 GiB
storage block
size:
512 bytes

CHAPTER

TWO

API

2.1 Initializing the libGUI

Initializing the libGUI is done with the following API:

```
#include "libgui.h"

void gui_init(uint16_t width,
              uint16_t height,
              cb_external_events external_events_cb);
```

The width and height specify the screen width and height in pixels. This defines the values of TILE_WIDTH_FULL and the number of tiles that can be packed on each others. The tile height TILE_HEIGHT_STD is calculated to be big enough to support a text content and a small icon of 45x45 pixels sized.

As holding graphical events (touch events) requires a blocking execution loop, external events (IPC, others) can't be handled in the same time. To resolve that, the libGUI proposes in the *gui_init()* last argument to declare a dedicated callbacks that is executed every graphical loop round.

This callback has the following API:

```
#include "libgui.h"

typedef void (*cb_external_events)(bool *refresh_gui_after);
```

This callback is executed each time the GUI loop is executed without graphical events (no touch detected). If this callback impacts the graphical state (tile modification or menu modification due to an external event like for e.g. an IPC), it should update the *refresh_gui_after* argument to **true**.

Hint: Avoid to write callbacks that may lock the CPU for too much time, this would freeze the GUI while the callback is being executed

---

2.2 Declaring menus

A graphical user interface is basisally composed of at least one menu.

A menu is declare using the following:

```
#include "libgui.h"

menu_desc_t main_menu;
ret = gui_declare_menu("MAIN", &main_menu);
if (ret != GUI_ERR_NONE) {
    printf("error while declaring menu: %d\n", ret);
}
```

Caution: All declaring functions and setters in libGUI return a gui_error_t return type, which can be one of GUI_ERROR_NONE, GUI_EROR_FULL (no more space), GUI_ERROR_INVAL (invalid parameters)

The menu name is for informatiion only, has the menu descriptor is the one used for any future menu update.

When all menus have been declared (or at least the main menu), the default menu must be defined to knwow which menu should be used to start the GUI main loop.

Warning: The default menu **must** be declared before executing the libgui main loop

Declaring the default menu is done using the corresponding menu descriptor:

```
# include "libgui.h"

menu_desc_t main_menu;

/* menu declaration */
ret = gui_declare_menu("MAIN", &main_menu);
if (ret != GUI_ERR_NONE) {
    printf("error while declaring menu: %d\n", ret);
}

/* set main menu as default */
gui_declare_default_menu(main_menu);
```

2.3 Declaring tiles

Declaring tiles works like declaring menus. Each tile is associated to a tile descriptor, which can be use at any time after the declaration in order to modify the tile's properties.

Here is a typical tile declaration:

```
#include "libgui.h"

#define TILE_STATUS_BG   .r = 53,  .g = 88,  .b = 157
#define TILE_FG          .r = 255, .g = 255, .b = 255

/* two menus here */
menu_desc_t main_menu;
menu_desc_t status_menu;

tile_desc_t main_status_tile;
```

(continues on next page)

(continued from previous page)

```
/* declaring main menu and status menu */
... /* see above... */

/* decilaring status tile */
{
    tile_colormap_t colormap[2] = {
        { TILE_STATUS_BG },
        { TILE_FG }
    };

    tile_text_t text = {
        .text = "status submenu",
        .align = TXT_ALIGN_CENTER
    };

    tile_icon_t icon = {
         .data = status,
         .size = sizeof(status)
    };

    action.type = TILE_ACTION_MENU;
    action.target.menu = status_menu;

    ret = gui_declare_tile(main_menu, colormap, TILE_WIDTH_FULL, TILE_HEIGHT_STD, &
↪action, &text, &icon, &main_status_tile);

    if (ret != GUI_ERR_NONE) {
        printf("error while declaring tile: %d\n", ret);
    }
}
```

Here we have defined a tile with an icon and a text content. When this tile is touched, the libGUI refresh the screen and load the status menu.

2.3.1 About tile actions

A title can be associated to three types of actions:

- TILE_ACTION_NONE
- TILE_ACTION_MENU
- TILE_ACTION_CB

TILE_ACTION_NONE means that no action is executed when the tile is touched. This is a typical use case for empty tiles, used as graphical separators.

TILE_ACTION_MENU change the current menu. The screen is refreshed, showing the menu targetted by the tile. When using the action, the *action.target.menu* must be set with the target menu descriptor value.

TILE_ACTION_CB executes the given callback when the tile is touched. This callback is declared in the *action.target.callback* field, which must be set.

Callbacks must respect the following API:

```
typedef void (*gui_callback_t)(tile_desc_t tile);
```

Warning: This callback is different from the *external events callback* and is executed as a trigger on touchscreen events

The callback knows which tile has been touched as it get back the tile descriptor as first argument. The callback may:

- execute non-graphical content (sending or receiving IPCs, updating another driver or service component)
- execute graphic content. In that later case, the callback should inform the GUI that a refresh is requested at the end of the callback execution, using the *gui_force_refresh()* API call.

Warning: gui_force_refresh() immediatly reload the current menu content on the screen. Any callback manipulating the screen content must finish the interaction with the user before executing gui_force_refresh()

Danger: When using IPC, be careful to avoid slowpaths that may be user-visible, as the GUI is frozzen during the overall callback execution

2.3.2 About tile text

A tile can have:

- No text at all. In that case, the text argument of the tile declaration should be null
- A text content. In that case, the text argument must hold a text conent including:
  - a string
  - a text alignment (TXT_ALIGN_LEFT, TXT_ALIGN_CENTER or TXT_ALIGN_LEFT)

Text informations are set in the tile_text_t structure that is passed to the tile declaration function. If the tile hold no text, the argument is null.

2.3.3 About tile icon

A tile can hold an icon. This icon is fixed to 45x45 pixels size. Icons must be in RLE (Run-Length Encoding) format. This format permit to highly compress basic images such as icons without loss.

The RLE converter is distributed in the libGUI sources, under the **tools/convert_logo.pl** file.

Best way to generate clean RLE images

- First, select your logo. Avoid to use complex figures, which may generate big header files.
- In your editor (for e.g. gimp) use an indexed colormap. Reducing the number of color to a reduced number also reduce the size of the icon. The usage of indexed colormap reduce the impact of the successive color approximation of the RLE converter
- Choose a reasonable number of colors in your colormap (from 2 to 5, 8...)
- Check that the icon correspond to what you want
- Update the icon size to 45x45 pixels. You can use the method you whish, while the result is based on this image size
- Export your logo in a PNG figure
- Execute *./tools/convert_logo.pl <your_image>*

Including your icon

The execution of the RLE converter generate the resulting C header in stdout. You can save the output in a C header file and rename the fields prefixes if you which (beware to keep the same suffixes).

Hint: Instead of renaming the prefix, you can properly name your PNG figure to directly generate the correct variables prefix for your header file

---

Danger: Take a great care to avoid too complex image or too big colormap. The resulting RLE image may be huge! Check the size of the generated header file

Now that your icon has been included in the sources of your application, you can declare it while declaring the corresponding tile. Given an icon named *lock.png*

```
#include "icons/lock.h"

[...]

tile_icon_t icon = {
     .data = lock,
     .size = sizeof(lock)
};

action.type = TILE_ACTION_CB;
action.target.callback = my_lock_callback;

ret = gui_declare_tile(main_menu, colormap, TILE_WIDTH_FULL, TILE_HEIGHT_STD, &action,
↪ 0, &icon, &main_status_tile);

if (ret != GUI_ERR_NONE) {
    printf("error while declaring tile: %d\n", ret);
}
```

2.4 Handling dynamicity

Graphical components dynamicity permit to modify the properties of tiles (menus properties can't be updated). As long as the GUI main loop is executed, it is possible to change any tile properties through:

- the external events callback
- any of the tiles callbacks

The following fields of a tile can be changed:

- the text field (modifying, removing or adding a text content)
- the icon field (modyfing, removing or adding an icon content)

Warning: If the modification is done in a tile of the current menu, the callback should request a menu refresh. In the case of external_events_callback, just update the *refresh_gui_after* argument. For tiles callback, call gui_force_refresh()

It is also possible to change the current menu as a result of a non-graphical event (e.g. a received IPC). This can be done, in the external events callback, through a call to *gui_set_menu()*

```
static menu_desc_t lock_menu;

/* initializing menus and tiles */
uint8_t init_gui(void) {
    [...]
}

/* handling various external events, asynchronously */
void my_external_event_callback(bool *refresh_gui)
{
    uint8_t ret:
    char    mybuf[4];
    [...]
    ret = sys_ipc(IPC_RECV_ASYNC, id_othertask, 4, &mybuf);
    if (ret == SYS_E_DONE) {
        if (mybuf[0] == REQ_GOTO_MENULOCK) {
           gui_set_menu(lock_menu);
           *refresh_gui = true;
        }
    }
}
```

2.5 Executing the main GUI loop

Executing the main loop is basically a while loop on GUI event, executing the *gui_get_events()* function. A basic usage is the following:

```
while (1) {
   gui_get_events();
}
```

2.6 Miscellaneous

The libGUI permits to temporary lock the touchscreen, avoiding any user interaction. This is done by calling the following function:

```
gui_lock_touch();
```

The touchscreen can then be unlocked by a call to:

```
gui_unlock_touch();
```

> **Warning:** Unlocking the touchscreen must be done through an external event handled by the external events callback, as no more touch event is receive

These two function permit to lock the screen during critical phases of the device execution. They can be executed in association with a dedicated lock menu which is only reachable through a call to gui_set_menu().

CHAPTER

THREE

FAQ

- **Is the libGUI responsible for tft and touch driver init ?**

No. This library is not responsible for the driver initialization as the libGUI has no early_init phase. The task is responsible for early initialize and initialize the TFT and Touch drivers, and associated devices (e.g. SPI bus).

- **Is the libGUI can print-out icons bigger or smaller than 45x45 ?**

Not this version. Although, if you which to print splash screens, you can directly call the driver primitive to print out an RLE image. The ili9341 driver support tft_rle_image() API which permit to print an RLE image on the screen.

- **What is the maximum number of menus ?**

The maximum number of menus is configureable in the libGUI dediated config entry. A reasonable value should be arround 10

- **What is the maximum number of tiles ?**

The maximum number of tiles is configureable in the libGUI dediated config entry. A reasonable value should be arround 30. Remeber that a tile structure is big and increasing the number of allowed tiles may impact the memory size of the generated application.