RTKIOT GUI Documentation

Release v0.0.0.1

RTKIOT GUI community

Mar 27, 2025

CONTENTS

Note: Realtek solution, user guide, study guide and other documents listed on this tutorial (collectively, “Documents”) are provided “as is” and with all faults. Customers agree to use any Documents solely for agreed purpose and subject to the terms of this Disclaimer.

CHAPTER

ONE

GET STARTED

1.1 Source Project Download

- Download on GitHub: https://github.com/realmcu/HoneyGUI
- Download on Gitee: https://gitee.com/realmcu/HoneyGUI

1.2 Description

HoneyGUI is a graphics display framework independently developed by Realtek. It is an open-source embedded graphical user interface (GUI) library specifically designed for resource-constrained microcontrollers and embedded systems. HoneyGUI is lightweight, feature-rich, and highly customizable, making it widely used in consumer electronics, home appliances, medical devices, and smartwatches.

As a comprehensive display framework, HoneyGUI not only includes Realtek's self-developed display engine but also supports direct calls to external APIs such as LVGL and ARM2D for application development. Additionally, HoneyGUI provides a PC-based simulation environment, allowing developers to quickly develop and debug applications without relying on embedded hardware platforms. Furthermore, HoneyGUI can be used in conjunction with Realtek's proprietary front-end design tool, *RVD*, to achieve visual programming.

Here are several common methods for APP development:

- Develop applications using the RealGUI display engine by calling C/C++ APIs.
- Directly call LVGL APIs to develop applications.
- Directly call ARM-2D APIs to develop applications.
- Front-end development using JavaScript and XML. It is recommended to use `RVisualDesigner` as a *PC* -based design tool for low-code development.

The *GUI* framework has good portability, which can run on a variety of chips and *OS*. PC Windows version is provided.

1.3 Software Architecture

1.4 Installation For Windows

1.4.1 Install Compiler

Download the MinGW-w64 toolchain, unzip it to drive C, and add it to the system environment variable `Path`.

1. MinGW-w64 Download
2. Unzip and copy to directory: `C:\mingw64`
3. Add a environment variable: `C:\mingw64\bin`:

- Open the Start Menu and search for `Advanced system setting`.
- Show `System Properties` and then go to the `Advanced` tab.
- Click on the `Environment Variables` button.
- In the `User variables` section, find and select the Path variable and click `Edit`.
- Click `New` and add `C:\mingw64\bin`.
- Click `OK` to close all dialogs.

1.4.2 Install Python

Python 3.9.7 is tested.

1.4.3 Install Scons

Open a CMD window and execute the following commands to install the Python scons library.

```
> pip install scons==4.4.0
```

After installing the `MinGW-w64` toolchain and `scons` library, you can launch the application in two ways: stratup by CMD or startup by GUI.

1.4.4 Startup by CMD (Scons)

Open a CMD window in the `HoneyGUI` or `gui` folder, and then run the following command to start the application.

```
> cd win32_sim
> scons
> cd ..
> .\win32_sim\gui.exe
```

The `scons` command to perform the build process and then execute `gui.exe` to run it.

1.4.5 Startup by CMD (CMake)

- Dependency Software

  CMake (tested with version 3.31.2): https://cmake.org/download/

  MinGW-w64: mentioned before

- Initialization: In the `HoneyGUI` folder

```
> cd win32_sim
> mkdir build
> cd build
> cmake -G "MinGW Makefiles" ..
```

- Compilation: In the `HoneyGUI/win32_sim/build` folder

```
> cmake -G "MinGW Makefiles" ..
> mingw32-make -j 32
```

- Configuration: In the `HoneyGUI/win32_sim/build` folder

```
> cmake --build . --target menuconfig
```

- Run: In the `HoneyGUI` folder

```
> .\win32_sim\gui.exe
```

1.4.6 Startup by VSCode

Install VSCode

- Download VSCode
- Install C/C++ plug-in

Open Project

- Click `HoneyGUI.code-workspace` file

Run Project

You can select the `Run and Debug` options after entering the vscode interface, and then click the `Run` button.

1.4.7 Display

Watch Project

The watchface is displayed in the window, and you can interact with it by swiping and long pressing.

Dashboard Project

The dashboard is displayed in the window.

CHAPTER

TWO

GUI APPLICATION

The GUI framework diagram is shown in the figure below:

Fig. 1: HoneyGUI Framework

- In each project, multiple applications can exist simultaneously, but only one application can be in a running state at any given time, while the other applications will be in a suspended state.
- When using different rendering engines, the upper-layer implementation of the application will vary.
- Each *APP* can create its own dedicated thread, or it may choose not to.
- The APP can be installed, opened, closed, uninstalled, and switched.
- The GUI_SERVER performs operations such as traversing widgets according to refresh instructions, rendering the frame buffer, executing trigger callbacks, and scheduling apps.

2.1 C-APP Application

- In this chapter, we will explore the creation and management of C-APPs within GUI framework. A C-APP is essentially an application that users can develop to craft interactive and visually appealing user interfaces. Each C-APP can be opened, closed, switched between, and can incorporate dynamic transition effects during switching.
- The displayed content within a C-APP is organized using a nested widget tree structure. This structure includes container widgets such as windows, scrollable pages, and switchable tabs, as well as content display widgets like text, images, and canvases.
- In addition to the default functions and effects, widgets within C-APPs offer a high degree of customization. Users can set up custom frame animations for widgets and bind events to execute their defined operations. This flexibility enables the creation of highly dynamic and interactive user interfaces tailored to specific needs and requirements.

2.1.1 Define A C-APP

- Define app handle using a specific name with `GUI_APP_DEFINE_NAME_ANIMATION` API:
- There are also other ways available to define the app:
  - `GUI_APP_DEFINE`
  - `GUI_APP_DEFINE_NAME`
  - `GUI_APP_DEFINE_NAME_ANIMATION_FUNC_CUSTOM`
  - `struct gui_app`

- Define app UI design entry function with `GUI_APP_ENTRY` API.
- The UI design entry function will be executed once when the app startup.

2.1.2 Create The Widget Tree of A C-APP

- This is a clock app, serving as an example for this section.
- In the image below, you can see that the app interface has options for a stopwatch and a countdown timer.
- Clicking on these options allows you to switch between them.

The graph below shows the widget tree structure simplified:

- SCREEN:APP_STOPWATCH: The main container for the stopwatch app.
  - WINDOW:LEFT_BUTTON: The window containing the left button.
    - CANVAS_RECT:LEFT_BUTTON: The background canvas of the left button.
    - TEXTBOX:LEFT_BUTTON: The text label for the left button.
  - WINDOW:RIGHT_BUTTON: The window containing the right button.
    - CANVAS_RECT:RIGHT_BUTTON: The background canvas of the right button.
    - TEXTBOX:RIGHT_BUTTON: The text label for the right button.
  - MULTI_LEVEL:0_0: A multi-level container.
    - MULTI_LEVEL:1_0: A sub-container within the multi-level container, for the stopwatch.
    - MULTI_LEVEL:1_1: Another sub-container within the multi-level container, for the countdown timer.

2.1.3 C-APP Operations

Here is how the earlier mentioned operations could be applied specifically to the Stopwatch app:

- `GUI_APP_SHUTDOWN(APP_STOPWATCH)` : This command will close the Stopwatch application. If the app is running a timer, it will stop the timer and close the interface. Any associated resources will be freed upon shutdown.
- `GUI_APP_STARTUP(APP_STOPWATCH)` : This command will initialize and start the Stopwatch application. The user interface will be displayed, and the app will be ready to start recording time.
- `GUI_APP_SWAP(APP_STOPWATCH, APP_MAP)` : This will switch from the currently running Stopwatch app to the Map app.

2.1.4 C-APP Transition Animation

C-APP provides a robust feature set for managing transition animations between applications. It offers three main functionalities: `built-in animations`, `custom animations`, and `layer management`. These features are designed to enhance the user experience by providing smooth and visually appealing transitions.

- Built-in Animations

  C-APP allows developers to easily implement built-in animations for app transitions using the `GUI_APP_DEFINE_NAME_ANIMATION` API. This API lets you specify the transition animations that occur when an app is opened or closed. The second parameter is used to define the animation for opening an app, while the third parameter specifies the animation for closing an app, such as `GUI_APP_ANIMATION_1`. This straightforward API streamlines the process of integrating transition effects within your applications.

- Custom Animations

  For more complex or unique animation requirements, C-APP supports custom animations through the `GUI_APP_DEFINE_NAME_ANIMATION_FUNC_CUSTOM` API. This feature enables developers to set custom animation callback functions for both opening and closing transitions. The second parameter is the callback function for the opening animation, and the third parameter is for the closing animation. These callback functions are defined using the `GUI_ANIMATION_CALLBACK_FUNCTION_DEFINE` API. This API provides an animation structure `gui_animate_t` instance as an argument, which includes members that offer insights into the progress and status of the animation, allowing for fine-tuned control and customization.

- Layer Management

  C-APP also includes APIs for managing the layering of applications, which can be crucial for visual hierarchy and user experience. The `gui_app_layer_top` and `gui_app_layer_bottom` APIs allow developers to define the layer relationship between the currently active app and the app that is about to open. This functionality ensures the correct ordering of windows and can help in maintaining the intended focus and organization of the app interfaces.

Example

- Built-in Animations
  - Define a C-APP
    - Startup Animation: Zoom In from Screen Center (`GUI_APP_ANIMATION_1`)
    - Shutdown Animation: Zoom Out to Screen Center (`GUI_APP_ANIMATION_5`)
  - Swap to the C-APP
    - From app watch to APP_STOPWATCH
- Custom Animations
  - Define a C-APP
    - Startup Animation: Pop-Up from Bottom of Screen (`heart_rate_startup`)
    - Shutdown Animation: Slide Down to Disappear (`heart_rate_shutdown`)
  - Swap to the C-APP
    - From app watch to APP_HEART_RATE

2.1.5 API

Warning: doxygenfile: Cannot find file “gui_app.h

2.2 Use LVGL Design An Application

2.2.1 LVGL introduction

- LVGL Website
- LVGL Document

- LVGL Intro

LVGL (Light and Versatile Graphics Library) is the most popular free and open-source embedded graphics library to create beautiful UIs for any MCU, MPU and display type. LVGL provides everything you need to create an embedded GUI with easy-to-use graphical elements, beautiful visual effects and a low memory footprint.

LVGL showcases Demo effects on its official website to demonstrate the UI building capabilities of LVGL. The online documentation serves as the primary development resource for LVGL, providing detailed information on the design and operational logic of LVGL, instructions on using various widgets, a wide range of example programs, and guidelines for porting LVGL. Whether you are a beginner or an experienced developer, you can quickly get started and gain a deep understanding of LVGL's functionality and features based on the online documentation.

- LVGL Demo
- LVGL Example

2.2.2 HoneyGUI Simulator

A simulator is a powerful tool used for developing UI that simulates the UI interface of embedded devices on a computer. It can mimic the behavior and appearance of a real hardware platform, providing developers with a convenient environment to quickly create, debug, and test UI designs.

The primary purpose of a simulator is to display and interactively test the designed UI interface in real-time, thereby reducing the time and cost of repetitive testing on actual hardware. By using a simulator, developers can iterate designs quickly, view the effects in real-time, and perform debugging and validation. This greatly speeds up UI development and improves workflow efficiency.

Using a simulator has the following advantages:

- Real-time preview: The simulator can show the immediate effects of the UI interface, allowing developers to quickly see the appearance and functionality of their design, facilitating adjustments and modifications.
- Cross-platform support: Simulators can run on computers, eliminating the need for specific hardware platforms.
- Time and resource-saving: Simulators help avoid frequent flashing and testing of UI on actual hardware, reducing additional time and cost overhead.
- Debugging and testing: Simulators provide rich debugging and testing capabilities to inspect the interaction, event handling, and layout effects of UI elements, aiding problem-solving and performance optimization.

Run LVGL in HoneyGUI Simulator

HoneyGUI Simulator is based on the scons tool and MinGW-w64 toolchain. It can be run and debugged in VScode. For specific environment setup and running instructions, please refer to the *Get Started* section.

After completing the environment setup for the HoneyGUI Simulator, when you start running it, you will see the default HoneyGUI project in the simulator. To modify the simulator configuration file to run an LVGL project, go to the path `your HoneyGUI dir/win32_sim/` and open the file `menu_config.h`, which is the configuration file for the simulator. Under the section **HoneyGUI Demo Select**, comment out all the demos. Under the section **HoneyGUI Enable LVGL**, enable `CONFIG_REALTEK_BUILD_LVGL_GUI`. Then, start running it again in VScode. After the build is successful, you will see the default LVGL demo project running in the simulator.

1. If you need to modify the screen size, open the file `SConscript` under the directory `your HoneyGUI dir/realgui/example/demo/`, and modify the values of `DRV_LCD_WIDTH` and `DRV_LCD_HEIGHT` to the desired pixel values.

HoneyGUI LVGL

The directories and files related to LVGL in HoneyGUI are as follows:

```
HoneyGUI Dir
|-- Arm2D
|-- cmake
|-- doc
|-- realgui
|   |-- 3rd
|   |-- app
|   |-- core
|   |-- dc
|   |-- engine
|   :
|   |__ example
|       |-- BAK
|       |-- demo
|       |   |__ app_ui_lvgl.c               // simulator LVGL UI entrance
|       :
|       :
|       |__ screen_lvgl
|           |-- assets                       // LVGL user image and font C files
|           |   |__ lvgl_example_assets.c    // assets example
|           |
|           |-- root                         // file system root folder
|           |-- _bin_mkromfs.py
|           |-- mkromfs_0x4600000.bat        // User Data pack script
|           |-- resource.h                   // resource files packed address
|           |__ root(0x4600000).bin          // packed User Data
|
|-- keil_sim
|-- lib
|-- lvgl_v8                                  // LVGL v8.3
|   |-- demos                                // LVGL demo source files
|   |   |-- benchmark
|   |   |-- keypad_encoder
|   |   |-- music
|   |   |-- stress
|   |   |__ widgets
|   |
|   |-- docs
|   |-- env_support
|   |-- examples                             // LVGL example source files
|   |   |-- anim
|   |   |-- arduino
|   |   |-- assets
|   |   |-- event
|   |   |-- get_started
|   |   |-- layouts
|   |   |-- libs
|   |   |-- others
|   |   |-- porting                          // LVGL porting template
|   |   |-- scroll
|   |   |-- styles
|   |   |__ widgets                          // LVGL example widges
|   |
|   |-- rlottie
```

(continues on next page)

(continued from previous page)

```
|   |-- scripts
|   |-- src
|   |   :
|   |   |-- widgets
|   |   |__ font                              // LVGL internal font
|   |
|   |__ tests
|
|-- lvgl_v9                                   // LVGL v9
:
:
|__ win32_sim
    :
    |__ port                                  // Simulator porting
        |-- realgui_port                      // Simulator HoneyGUI porting
        |-- lvgl_port                         // Simulator LVGLv8 porting
        |    |-- lv_conf.h                    // Simulator LVGL configuration
        |    |-- lv_port_disp.c
        |    |-- lv_port_disp.h
        |    |-- lv_port_fs.c
        |    |-- lv_port_fs.h
        |    |-- lv_port_indev.c
        |    |__ lv_port_indev.h
        |
        |__ lvglv9_port                        // Simulator LVGLv9 porting
```

1. In HoneyGUI, the LVGL source files are located in the directory `your HoneyGUI dir/lvgl`:
   - demos: Contains various comprehensive built-in examples of LVGL. Some examples can be experienced on LVGL Demo .
   - docs: Contains the development documentation for LVGL. It can be read online on the LVGL documentation site: LVGL Document.
   - env_support: Provides support for various environments or platforms.
   - examples: Stores the built-in examples of LVGL. They can be experienced on LVGL Example .
   - scripts: Contains some processing scripts that are not typically used when using LVGL.
   - src: Stores the actual source code of LVGL. When developing with LVGL, the code files from this directory are used.
   - tests: Contains some CI testing files that are not used when using LVGL.
2. When running LVGL with the HoneyGUI simulator, the LVGL UI will start running from the file `app_ui_lvgl.c` under the directory `your HoneyGUI dir/realgui/example/demo`.
3. When running LVGL with the HoneyGUI simulator, the root directory pointed to by the LVGL file system interface is `your HoneyGUI dir/realgui/example/screen_lvgl/root/`.

2.2.3 Porting

- Documentation: LVGL Porting

LVGL provides extensive porting support, allowing developers to easily integrate it into various embedded systems and platforms. It supports drivers for various display devices, touchscreens, input devices, and custom GPUs. Developers can configure the porting according to the requirements of their projects, such as adjusting the display parameters when changing display devices, or adapting the input interface when replacing input devices. This article focuses on the porting process and methods for display devices, input devices, and file systems. For more details, please refer to LVGL Porting.

Note: The following examples do not include the specific implementation of hardware device drivers. They only illustrate how to integrate drivers with the LVGL interface. When implementing hardware device drivers, developers can complete the driver functionality under a consistent API framework with the example driver, in order to interface with the HoneyGUI driver layer. The porting interfaces of the example projects can be reused in higher layers.

Display

- Documentation: LVGL Porting Display, LVGL Overview Display

Once the developers have completed the debugging of the display device driver, and the device can communicate properly with the display device and show colors. This section explains how to interface the driver with LVGL's display interface to render LVGL's UI."

The display interface of LVGL is implemented in the file `lv_port_disp.c`. Display parameters are configured in the initialization function `void lv_port_disp_init(void)()`, such as screen size and frame buffer configuration. The display refresh function is defined as `void disp_flush(lv_disp_drv_t *disp_drv, const lv_area_t *area, lv_color_t *color_p)()`.

The file `lv_port_disp.c` has been configured with different rendering and screen-pushing methods for reference. Configure `DISPLAY_FLUSH_TYPE` to switch modes, where `RAMLESS_XXX` is suitable for display ICs without RAM, `RAM_XXX` is suitable for display ICs with RAM, `XXX_FULL_SCREEN_XXX` indicates pushing the entire screen each time, and `XXX_TWO_SEC` indicates rendering only the changed display content, with the unit being the size of two buffers. The pixel height of the buffer is defined by `SECTION_HEIGHT`.

For detailed display device porting methods and considerations, please refer to the documentation LVGL Porting Display. The following code snippet demonstrates porting a display IC without RAM:

- When using a display IC without RAM, a frame buffer that covers the entire screen size needs to be allocated. Therefore, two frame buffers with a size equal to the screen size are allocated on the PSRAM for display. The macro definitions for display parameters are defined in the file `lv_conf.h`.
- If the display IC used has RAM, the size of the frame buffer does not need to be the same as the screen size. Due to different screen update methods, the `LVGL_USE_EDPI` in `lv_port_disp.c` needs to be configured as not enabled (0) to switch the `disp_flush()` function for screen update adaptation.

```
// flush func 1
#define RAMLESS_TWO_FULL_SCREEN         0  // double buffer, full refresh

// flush func 2
#define RAM_TWO_FULL_SCREEN_NO_SEC      1  // double buffer, full refresh
#define RAM_ONE_FULL_SCREEN_TWO_SEC     2  // two buffer
#define RAM_DIRECT_TWO_SEC              3  // two buffer


// two buffer: section height
```

(continues on next page)

(continued from previous page)

```
#define SECTION_HEIGHT                  40


#define DISPLAY_FLUSH_TYPE              RAMLESS_TWO_FULL_SCREEN

#if (DISPLAY_FLUSH_TYPE == RAMLESS_TWO_FULL_SCREEN)
#define LVGL_USE_EDPI       1
#else
#define LVGL_USE_EDPI       0
#endif


// frame buffer config
#define LV_PORT_BUF1        (uint32_t)0x08000000   // address in PSRAM
#define LV_PORT_BUF2        (uint32_t)(0x08000000 + MY_DISP_HOR_RES * MY_DISP_VER_RES␣
↪* LV_COLOR_DEPTH / 8)

void lv_port_disp_init(void)
{
    /*-------------------------
    * Initialize your display
    * -----------------------*/
    disp_init();

    /*-----------------------------------
    * Register the display in LVGL
    *----------------------------------*/

    static lv_disp_drv_t disp_drv;                  /*Descriptor of a display driver*/
    lv_disp_drv_init(&disp_drv);                    /*Basic initialization*/

    /*Set up the functions to access to your display*/

    /*Set the resolution of the display*/
    disp_drv.hor_res = MY_DISP_HOR_RES;
    disp_drv.ver_res = MY_DISP_VER_RES;

    /*Used to copy the buffer's content to the display*/
    disp_drv.flush_cb = disp_flush;

    /*-----------------------------
    * Create a buffer for drawing
    *----------------------------*/

    /**
    * LVGL requires a buffer where it internally draws the widgets.
    * Later this buffer will passed to your display driver's `flush_cb` to copy its␣
↪content to your display.
    * The buffer has to be greater than 1 display row
    *
    * There are 3 buffering configurations:
    * 1. Create ONE buffer:
    *      LVGL will draw the display's content here and writes it to your display
    *
    * 2. Create TWO buffer:
    *      LVGL will draw the display's content to a buffer and writes it your␣
↪display.
```

(continues on next page)

(continued from previous page)

```
    *      You should use DMA to write the buffer's content to the display.
    *      It will enable LVGL to draw the next part of the screen to the other␣
↪buffer while
    *      the data is being sent form the first buffer. It makes rendering and␣
↪flushing parallel.
    *
    * 3. Double buffering
    *      Set 2 screens sized buffers and set disp_drv.full_refresh = 1.
    *      This way LVGL will always provide the whole rendered screen in `flush_cb`
    *      and you only need to change the frame buffer's address.
    */
#if (DISPLAY_FLUSH_TYPE == RAMLESS_TWO_FULL_SCREEN || DISPLAY_FLUSH_TYPE == RAM_TWO_
↪FULL_SCREEN_NO_SEC)
    static lv_disp_draw_buf_t draw_buf_dsc_3;
    lv_color_t *buf_3_1 = (lv_color_t *)LV_PORT_BUF1;           /*A screen sized␣
↪buffer*/
    lv_color_t *buf_3_2 = (lv_color_t *)LV_PORT_BUF2;           /*Another screen␣
↪sized buffer*/
    lv_disp_draw_buf_init(&draw_buf_dsc_3, buf_3_1, buf_3_2,
                          MY_DISP_VER_RES * MY_DISP_HOR_RES);   /*Initialize the␣
↪display buffer*/

    /*Set a display buffer*/
    disp_drv.draw_buf = &draw_buf_dsc_3;

    /*Required for Example 3)*/
    disp_drv.full_refresh = 1;

#elif (DISPLAY_FLUSH_TYPE == RAM_DIRECT_TWO_SEC || DISPLAY_FLUSH_TYPE == RAM_ONE_FULL_
↪SCREEN_TWO_SEC)
#if 1
    static uint8_t __attribute__((aligned(4))) disp_buff1[MY_DISP_HOR_RES * SECTION_
↪HEIGHT *
                                                                                LV_COLOR_
↪DEPTH / 8];
    static uint8_t __attribute__((aligned(4))) disp_buff2[MY_DISP_HOR_RES * SECTION_
↪HEIGHT *
                                                                                LV_COLOR_
↪DEPTH / 8];
#else
    uint8_t *disp_buff1 = lv_mem_alloc(MY_DISP_HOR_RES * SECTION_HEIGHT * LV_COLOR_
↪DEPTH / 8);
    uint8_t *disp_buff2 = lv_mem_alloc(MY_DISP_HOR_RES * SECTION_HEIGHT * LV_COLOR_
↪DEPTH / 8);
#endif
    static lv_disp_draw_buf_t draw_buf_dsc_2;
    lv_color_t *buf_2_1 = (lv_color_t *)disp_buff1;
    lv_color_t *buf_2_2 = (lv_color_t *)disp_buff2;

    if (!buf_2_1 || !buf_2_2)
    {
        DBG_DIRECT("LVGL frame buffer is NULL");
        while (1);
    }
    lv_disp_draw_buf_init(&draw_buf_dsc_2, buf_2_1, buf_2_2,
                          MY_DISP_HOR_RES * SECTION_HEIGHT);   /*Initialize the display␣
↪buffer*/
```

(continues on next page)

(continued from previous page)

```
    /*Set a display buffer*/
    disp_drv.draw_buf = &draw_buf_dsc_2;

    /*Required for Example 2)*/
    disp_drv.full_refresh = 0;

    // disp_drv.rounder_cb = rounder_cb;

#endif

    /*Finally register the driver*/
    lv_disp_drv_register(&disp_drv);
}
```

Input Device

- Documentation: LVGL Porting Input devices

Once the developers have completed the debugging of the input device driver, and the device can communicate properly with the input device. This section explains how to interface the driver with LVGL's input interface to interact with LVGL's UI.

The input interface of LVGL is implemented in the file `lv_port_indev.c`. Input device parameters are configured in the initialization function `void lv_port_indev_init(void)()`, including selecting the device type, etc. The input data acquisition function is configured in the function pointer `indev_drv.read_cb()`, which depends on the type of input device and is integrated in `lv_port_indev.c`.

For detailed input device porting methods and considerations, please refer to the documentation LVGL Porting Input devices. The following code snippet demonstrates porting a touch IC:

- In the initialization function `void lv_port_indev_init(void)()`, select and register the corresponding type of input device. For example, for a touchpad device, select **Touchpad**.
- LVGL will retrieve the input data through the function pointer `indev_drv.read_cb()`. Developers need to provide the input data in the function it points to. For a touch screen device, it would be the function `void touchpad_read(lv_indev_drv_t *indev_drv, lv_indev_data_t *data)()`. For a touch screen input device, you only need to provide the coordinates of the touch point and the touch state.

```
void lv_port_indev_init(void)
{
    /**
    * Here you will find example implementation of input devices supported by␣
↪LittelvGL:
    *  - Touchpad
    *  - Mouse (with cursor support)
    *  - Keypad (supports GUI usage only with key)
    *  - Encoder (supports GUI usage only with: left, right, push)
    *  - Button (external buttons to press points on the screen)
    *
    *  The `..._read()` function are only examples.
    *  You should shape them according to your hardware
    */

    static lv_indev_drv_t indev_drv;
```

(continues on next page)

(continued from previous page)

```
    /*------------------
    * Touchpad
    * ------------------*/

    /*Initialize your touchpad if you have*/
    touchpad_init();

    /*Register a touchpad input device*/
    lv_indev_drv_init(&indev_drv);
    indev_drv.type = LV_INDEV_TYPE_POINTER;
    indev_drv.read_cb = touchpad_read;
    indev_touchpad = lv_indev_drv_register(&indev_drv);
}


/*------------------
* Touchpad
* ------------------*/

static uint16_t touch_x = 0;
static uint16_t touch_y = 0;
static bool touch_pressing = 0;


/*Initialize your touchpad*/
static void touchpad_init(void)
{
    /*Your code comes here*/
}

/*Will be called by the library to read the touchpad*/
static void touchpad_read(lv_indev_drv_t *indev_drv, lv_indev_data_t *data)
{
    static lv_coord_t last_x = 0;
    static lv_coord_t last_y = 0;

    /* rt touch read port */
    if (drv_touch_read(&touch_x, &touch_y, &touch_pressing) == false)
    {
        return;
    }

    /*Save the pressed coordinates and the state*/
    if (touchpad_is_pressed())
    {
        touchpad_get_xy(&last_x, &last_y);
        data->state = LV_INDEV_STATE_PR;
    }
    else
    {
        data->state = LV_INDEV_STATE_REL;
    }

    /*Set the last pressed coordinates*/
    data->point.x = last_x;
    data->point.y = last_y;
}
```

(continues on next page)

(continued from previous page)

```
/*Return true is the touchpad is pressed*/
// static lv_coord_t touch_x;
// static lv_coord_t touch_y;
static bool touchpad_is_pressed(void)
{
    /*Your code comes here*/
    return touch_pressing;

}

/*Get the x and y coordinates if the touchpad is pressed*/
static void touchpad_get_xy(lv_coord_t *x, lv_coord_t *y)
{
    /*Your code comes here*/
    (*x) = touch_x;
    (*y) = touch_y;
}
```

File System

- Documentation: LVGL Overview File system

Using a file system to manage storage media makes data more organized and easier to maintain. It can improve compatibility and cross-platform support for external storage devices. Through the file system interface, developers can easily manipulate file data, making it more flexible and efficient. Integrating the file system with LVGL allows resource data to be stored separately from project code, reducing compilation time, improving development efficiency, and enhancing the flexibility of UI design.

The file system interface of LVGL is implemented in the file `lv_port_fs.c`. The file system is configured in the initialization function `void lv_port_fs_init(void)()`, which includes initializing the file system and mounting drive letters. Developers need to integrate the interfaces of various file system functions into the corresponding LVGL fs porting functions, ensuring that the input and output data formats are consistent with the interface definitions.

For detailed file system porting methods and considerations, please refer to the documentation LVGL Overview File system. The following example demonstrates the porting of **ROMFS**.

Note: ROMFS is a read-only file system, thus it does not support file writing.

```
#include "romfs.h"

/**********************
*      MACROS
**********************/
#define ROMFS_ADDR 0x04600000
/**********************
*   GLOBAL FUNCTIONS
**********************/

void lv_port_fs_init(void)
{
    /*-------------------------------------------------
    * Initialize your storage device and File System
    * -------------------------------------------------*/
    fs_init();
```

(continues on next page)

(continued from previous page)

```
    /*---------------------------------------------------
    * Register the file system interface in LVGL
    *--------------------------------------------------*/

    /*Add a simple drive to open images*/
    static lv_fs_drv_t fs_drv;
    lv_fs_drv_init(&fs_drv);

    /*Set up fields...*/
    fs_drv.letter = 'F';
    fs_drv.open_cb = fs_open;
    fs_drv.close_cb = fs_close;
    fs_drv.read_cb = fs_read;
    fs_drv.write_cb = fs_write;
    fs_drv.seek_cb = fs_seek;
    fs_drv.tell_cb = fs_tell;

    fs_drv.dir_close_cb = fs_dir_close;
    fs_drv.dir_open_cb = fs_dir_open;
    fs_drv.dir_read_cb = fs_dir_read;

    lv_fs_drv_register(&fs_drv);
}

/**********************
*   STATIC FUNCTIONS
**********************/

/*Initialize your Storage device and File system.*/
static void fs_init(void)
{
    /*E.g. for FatFS initialize the SD card and FatFS itself*/

    /*You code here*/
    romfs_mount((void *)ROMFS_ADDR);
}

/**
* Open a file
* @param drv       pointer to a driver where this function belongs
* @param path      path to the file beginning with the driver letter (e.g. S:/folder/
↪file.txt)
* @param mode      read: FS_MODE_RD, write: FS_MODE_WR, both: FS_MODE_RD | FS_MODE_WR
* @return          a file descriptor or NULL on error
*/
static void *fs_open(lv_fs_drv_t *drv, const char *path, lv_fs_mode_t mode)
{
    lv_fs_res_t res = LV_FS_RES_NOT_IMP;

    void *f = NULL;

    if (mode == LV_FS_MODE_WR)
    {
        /*Open a file for write*/
        f = NULL;         /*Add your code here*/
    }
```

(continues on next page)

(continued from previous page)

```
    else if (mode == LV_FS_MODE_RD)
    {
        /*Open a file for read*/
        const char *filePath = path;
        f = (void *)open(filePath, O_RDONLY);        /*Add your code here*/
    }
    else if (mode == (LV_FS_MODE_WR | LV_FS_MODE_RD))
    {
        /*Open a file for read and write*/
        f = NULL;         /*Add your code here*/
    }

    return f;
}

/**
* Close an opened file
* @param drv       pointer to a driver where this function belongs
* @param file_p    pointer to a file_t variable. (opened with fs_open)
* @return          LV_FS_RES_OK: no error or  any error from @lv_fs_res_t enum
*/
static lv_fs_res_t fs_close(lv_fs_drv_t *drv, void *file_p)
{
    lv_fs_res_t res = LV_FS_RES_NOT_IMP;

    /*Add your code here*/
    res = close((int)file_p);
    return res;
}

/**
* Read data from an opened file
* @param drv       pointer to a driver where this function belongs
* @param file_p    pointer to a file_t variable.
* @param buf       pointer to a memory block where to store the read data
* @param btr       number of Bytes To Read
* @param br        the real number of read bytes (Byte Read)
* @return          LV_FS_RES_OK: no error or  any error from @lv_fs_res_t enum
*/
static lv_fs_res_t fs_read(lv_fs_drv_t *drv, void *file_p, void *buf, uint32_t btr,␣
↪uint32_t *br)
{
    lv_fs_res_t res = LV_FS_RES_OK;

    /*Add your code here*/
    *br = read((int)file_p, buf, btr);
    return res;
}
```

ROMFS File System Image

HoneyGUI provides support for packaging *ROMFS* file system images:

1. The working directory is `your HoneyGUI dir/realgui/example/screen_lvgl/`. The packaging process requires Python environment support. The external file resources used in the project need to be packaged as a file system image and downloaded as *User Data*.
2. Open the working directory and place the files to be packaged in the `root/` folder. Double-click the `mkromfs_0x4600000.bat` script to generate the file system image `root(0x4600000).bin` and the resource mapping address `resource.h`. The default *base address* of the files is *0x4600000*. `resource.h` records the mapping address of the packaged files. Since *ROMFS* supports direct access using physical addresses, developers can access the resource files directly through the mapping address.
3. Use the *User Data* feature of the MP Tool to download and burn the file system image to flash. The burn address should match the *base address*. If you need to modify the *base address*, you can modify the **"–addr <number>"** parameter in the `mkromfs_0x4600000.bat` script. For example, the following example changes the *base address* from *0x4600000* to *0x4000000*.

```
# before - base address: 0x4600000, image: root(0x4600000).bin
python _bin_mkromfs.py --binary --addr 0x4600000 root root(0x4600000).bin


# after  - base address: 0x4000000, image: root(0x4000000).bin
python _bin_mkromfs.py --binary --addr 0x4000000 root root(0x4000000).bin
```

Note:

1. This packaging tool is only applicable for creating filesystem images of ROMFS.
2. The packaging process is not a simple concatenation of files; it also records the directory information and file details of the filesystem.

LittleFS File System Image

The LittleFS file system supports read and write operations and features power-loss protection. HoneyGUI provides packaging support for LittleFS file system images:

1. The working directory is `your HoneyGUI dir/realgui/example/screen_lvgl/root_lfs`. External file resources used by the project will be packaged into a file system image and ultimately downloaded as *User Data*.
2. Open the working directory and place the files you need to package under the `root/` folder. Double-click the script `mklittlefs_img.bat` to generate the file system image `root.bin`.
3. Use the *User Data* function in MP Tool to download and write the file system image to flash. To change the size of the file system, modify the **"-s <number>"** parameter in the script `mklittlefs_img.bat`. When using interfaces from `rtk_fs.c` for file operations, ensure that `RTK_FS_MNT_ADDR` matches the write address, and `MAX_LFS_SIZE` matches the file system size.
4. If you need to unpack a file system image, double-click the script `unpack_littlefs_img.bat` to unpack `root.bin` into the `root_up/` folder.

```
# pack image:
# -c <pack_dir>,  --create <pack_dir>
```

(continues on next page)

(continued from previous page)

```
# create littlefs image from a directory
#
# -b <number>,  --block <number>
# fs block size, in bytes
#
# -p <number>,  --page <number>
# fs page size, in bytes
#
# -s <number>,  --size <number>
# fs image size, in bytes

mklittlefs.exe -c root/  root.bin  -b 4096  -s 512000 -p 16


# unpack image:
# -l,  --list
# list files in littlefs image
#
# -u <dest_dir>,  --unpack <dest_dir>
# unpack littlefs image to a directory

mklittlefs.exe root.bin  -l
mklittlefs.exe root.bin  -u root_up/
```

Note:

1. This packaging tool is only applicable for creating filesystem images of LittleFS.

2.2.4 LVGL Benchmark

LVGL Benchmark is a performance testing tool designed to evaluate the graphical display performance of the LVGL library in various hardware and software environments. By running the Benchmark, users can obtain data on frame rate, rendering speed, and memory usage, helping to optimize display configurations and debug performance issues. The Benchmark includes various test scenarios such as graphical drawing, animations, and text rendering, each simulating common operations in real applications. Users can use these tests to compare the performance of different configurations and platforms, enabling targeted optimization adjustments. The official documentation for the LVGL benchmark test is located at `your HoneyGUI dir/lvgl/demos/benchmark/README.md`.

Benchmark for Reference

Table 1: Benchmark Result

Chip Model	CPU CLK	Accelerator	Display Size	Buffering Configurations	Result
RTL8762E	40MHz	SW	240*280	Double buffing	Weighted FPS:15; Opa. speed: 100%
RTL8762E	40MHz	SW	80*160	Double buffing	Weighted FPS:34; Opa. speed: 95%
RTL8762D	90MHz	SW	240*280	Double buffing	Weighted FPS:161; Opa. speed: 77%
RTL8762D	90MHz	SW	80*160	Double buffing	Weighted FPS:337; Opa. speed: 95%
RTL8772G	125MHz	PPE1.0	480*480	Two buffer	Weighted FPS:20; Opa. speed: 100%
RTL8772G	125MHz	PPE1.0	240*280	Double buffing	Weighted FPS:721; Opa. speed: 77%
RTL8773E	100MHz	PPE2.0	390*450	Double buffing	Weighted FPS:159; Opa. speed: 86%

Table 2: Render acceleration on different platforms

Chip Model	CPU CLK	Hardware Accelerator	Image Rendering	Image Transparency	Image Scaling	Image Rotation	Rounded Rectangle	Rectangle Filling	RLE Decoding	Characters	Lines
RTL87	125M	PPE1.0	HW	HW	HW	SW	SW+HW	HW	HW	SW	SW
RTL87	100M	PPE2.0	HW	HW	HW	HW	SW+HW	HW	HW	SW	SW

Note:

1. Effects involving LVGL Mask require SW processing.
2. RTL8772G supports the Helium hardware accelerator.

2.2.5 Start with Demo

- LVGL Demo
- LVGL Example

It is recommended for developers to read and understand the LVGL Overview and LVGL Widgets - Base object sections before starting development. This will help them grasp the design concepts and logic of LVGL.

LVGL provides a rich set of demos and examples to help developers understand and familiarize themselves with the usage of various widgets and features.

- The LVGL Demo showcases comprehensive demos with their source code stored in the directory `your HoneyGUI dir/lvgl/src/demo`. Developers can directly invoke the corresponding `lv_demo_xxx()` function to explore and understand them.

- The online documentation LVGL Example demonstrates the running effects of various examples, with their source code stored in the directory `your HoneyGUI dir/lvgl/src/example`. Developers can directly call the corresponding `lv_example_xxx()` function to familiarize themselves with widgets and understand their features.

2.2.6 Resource Converter

To use images and fonts in LVGL, they need to be converted to formats that LVGL can recognize using specific tools. LVGL supports converting resources to C array format and bin binary file format.

In the C array format, the resources will be included in the compilation process. They will be compiled every time the program logic changes, and the size of the resources will be included in the APP image.

In the bin binary file format, the resources are not included in the compilation. They are stored separately and require a file system or other means to access them. An example `lvgl_example_assets.c` is provided in the path `your HoneyGUI dir/realgui/example/screen_lvgl/assets/` to demonstrate how to configure resources of different formats for the widgets.

Image Converter

LVGL Image Converter

- Online conversion tool: LVGL Image Converter
- Documentation: LVGL Overview Images

Please refer to the following steps for usage in LVGL Overview Images - Online Converter:

1. Select the LVGL version.
2. Choose the image file.
3. Select the color format for the output file.

   For color format details, please refer to LVGL Overview Images - color format.
4. Choose the type of output image (C array/binary file).
5. Click *Convert* to obtain the output file.

The LVGL Overview Images document provides detailed instructions on how to use image resources and the image conversion tool in LVGL, along with simple usage examples. To automatically build image resources generated as C arrays, place them under the directory `your HoneyGUI dir/realgui/example/screen_lvgl/assets/` directory.

It's worth mentioning that when using the bin file as an image resource, the data in the bin file follows the format of `4 Byte header + data`. The `lv_img_header_t` contains information such as `color format`, `width`, and `height`. To construct a complete `lv_img_dsc_t` to describe the image, you can calculate the `data_size` using the information from the `lv_img_header_t`.

```
typedef struct {

    uint32_t cf : 5;          /*Color format: See `lv_img_color_format_t`*/
    uint32_t always_zero : 3; /*It the upper bits of the first byte. Always zero to␣
↪look like a
                               non-printable character*/

    uint32_t reserved : 2; /*Reserved to be used later*/
```

(continues on next page)

(continued from previous page)

```
    uint32_t w : 11; /*Width of the image map*/
    uint32_t h : 11; /*Height of the image map*/
} lv_img_header_t;

/** Image header it is compatible with
* the result from image converter utility*/
typedef struct {
    lv_img_header_t header; /**< A header describing the basics of the image*/
    uint32_t data_size;     /**< Size of the image in bytes*/
    const uint8_t * data;   /**< Pointer to the data of the image*/
} lv_img_dsc_t;
```

HoneyGUI Image Convert Tool

- Download link for the conversion tool: HoneyGUI Image Convert Tool
- Documentation: HoneyGUI Image Convert Tool - Doc

When further compression of image resource space is needed, the HoneyGUI Image Convert Tool supports compressing and converting images. The IC supports both software and hardware decoding. The HoneyGUI Image Convert Tool uses RLE (Run-length Encoding) compression, a simple lossless algorithm that reduces storage space by encoding consecutive repeated pixel values and the number of repetitions. It has low computational complexity and high compression rates, making it ideal for compressing GUI resources.

Compressing Images

Users can utilize the HoneyGUI Image Convert Tool to convert image resources into RLE-compressed binary file format. For detailed usage steps, please refer to HoneyGUI Image Converter - Doc:

1. Select the image file to be compressed (supports PNG, JPEG, etc.)
2. Configure the image conversion parameters: enable *Compress*, choose *Compress Mode* as *RLE*, enable *Color Head*, and select *Color Space* as needed
3. Click to *Convert* and generate a compressed binary file

Importing into LVGL

The binary files generated by the HoneyGUI Image Convert Tool can be imported into LVGL for use:

1. If importing as a file

   Note: Modify the file extension to **.rle**, then place it into the file system at `your HoneyGUI dir/realgui/example/screen_lvgl/root`

   ```
   // file: lvgl_example_assets.c
   void load_img_rle_file(void)
   {
       lv_obj_t *icon = lv_img_create(lv_scr_act());
       lv_img_set_src(icon, "F:/logo_lvgl.rle");
       lv_obj_set_pos(icon, 0, 0);
   }
   ```

Note: When using RLE + ROMFS, the decoder will directly retrieve images from the file system, i.e. FLASH, without additional caching. For situations that require caching processing, please read file from filesystem to memory, and use it as a array.

2. If imported as a C array format

   - a. Open the LVGL image conversion tool and upload the compressed file to be converted, please refer to LVGL Image Converter.
   - b. In the *Color format* option, be sure to select **CF_RAW**
   - c. Export the converted image file as a C file, for example, `logo_lvgl_rle.c`

   Note1: The storage path of the converted file: Place the converted C file in the following reference path: `your HoneyGUI dir/realgui/example/screen_lvgl/assets`

   Note2: Modify the color format (cf) in the image descriptor: The exported C file, for example `logo_lvgl_rle.c`, needs to be modified to ensure :c:var:cf: `LV_IMG_CF_RAW`:

```
// file:logo_lvgl_rle.c
const lv_img_dsc_t logo_lvgl_rle = {
    .header.cf = LV_IMG_CF_RAW,
    .header.always_zero = 0,
    .header.reserved = 0,
    .header.w = 0,
    .header.h = 0,
    .data_size = 1889,
    .data = logo_lvgl_rle_map,
};
```

   - d. Include the generated C file in your project and create the image object:

```
// file:lvgl_example_assets.c
void load_img_rle_c_file(void)
{
    LV_IMG_DECLARE(logo_lvgl_rle);
    lv_obj_t *icon = lv_img_create(lv_scr_act());
    lv_img_set_src(icon, &logo_lvgl_rle);
    lv_obj_set_pos(icon, 0, 0);
}
```

3. If importing as a file, accessing image resources using file addresses

   - a. Construct the LVGL image header `lv_img_dsc_t`, for example:

```
// file: lvgl_example_assets.c
#include "resource.h"

const lv_img_dsc_t lvgl_test_img_rle = {
    .header.cf = LV_IMG_CF_RAW,
    .header.always_zero = 0,
    .header.reserved = 0,
    .header.w = 0,
    .header.h = 0,
    .data_size = 0,
    .data = LOGO_LVGL_RLE,
};
```

   Note: Set the color format in the image descriptor to cf = LV_IMG_CF_RAW

   - b. Access the image resources and create the widget:

```
// file: lvgl_example_assets.c
void load_img_rle_dataAddr_file(void)
{
    lv_obj_t *icon = lv_img_create(lv_scr_act());
    lv_img_set_src(icon, &lvgl_test_img_rle);
    lv_obj_set_pos(icon, 0, 0);
}
```

Enabling RLE Decoder in LVGL

To decode RLE compressed image resources in LVGL, you need to enable the RLE decoder and allocate cache space for it.

1. Enable the RLE decoder: in the configuration file `lv_conf.h`, locate the `LV_USE_RTK_IDU` macro definition and set it to enable (1)
2. Allocate decoding cache: Configure the following parameters in the `lv_conf.h` file:

- `LV_SSRAM_START`: The starting address of the cache
- `LV_SSRAM_SIZE`: Cache space size, ensuring that this size is sufficient to accommodate the decoding data of the largest entire image used

```
// file: lv_conf.h

/*RTK_IDU decoder library*/
#define LV_USE_RTK_IDU 1

#ifdef LV_USE_RTK_IDU
#define LV_MEM_PSRAM_ADR    0x08000000
#define LV_PSRAM_SIZE       (MY_DISP_HOR_RES * MY_DISP_VER_RES * 4)
#define LV_PSRAM_START      (LV_MEM_PSRAM_ADR + 2 * MY_DISP_HOR_RES * MY_DISP_VER_RES␣
↪* LV_COLOR_DEPTH / 8)
#ifndef LV_MEM_ADR
#define LV_MEM_ADR LV_PSRAM_START
#endif
#endif
```

Note: When using the RLE decoder along with ROMFS, the decoder will directly obtain images from the file system, i.e., FLASH, without additional caching.

Font Converter

- Online conversion tool: LVGL Font Converter
- Documentation: LVGL Overview Fonts

Please refer to the following steps for usage in LVGL Overview Font - Add a new font :

1. Set the name of the output font.
2. Set the height of the font in pixels.
3. Set the bpp (bits per pixel) of the font.

It represents how many bits are used to describe each pixel. Higher values result in better anti-aliasing and smoother edges, but larger font file size.

4. Choose the type of output font (C array/bin file).
5. Select the font file (TTF/WOFF).
6. Set the Unicode range of characters to convert, or directly list the characters that need to be converted.

The LVGL Overview Fonts document provides detailed instructions on how to use font resources and the font conversion tool in LVGL, along with simple usage examples. In the example, `lv_example_label_3()` demonstrates how to configure a specific font for a label widget. To automatically build font resources generated as C arrays, place them under the directory `your HoneyGUI dir/realgui/example/screen_lvgl/assets/` directory.

LVGL provides built-in fonts, which are saved as arrays in the directory `your HoneyGUI dir/lvgl/src/font/` . Each font file specifies the included characters at the beginning of the file. The built-in fonts include a Chinese font, `lv_font_simsun_16_cjk.c`, which is a CJK (Chinese, Japanese, and Korean) 16px font, but it is a single font size with a limited character set.

2.2.7 Development Resources

Online Doucument

- LVGL Document

The LVGL Document provides comprehensive technical documentation and tutorials to help developers better understand and use the LVGL graphics library. The documentation includes the following:

- Overview and features: The documentation introduces the basic concepts and features of LVGL, including graphical objects, screen management, event handling, theme styles, and more. Users can read the documentation to understand the core functionality and advantages of LVGL.
- Application development guide: The documentation provides detailed application development guides, including how to initialize and configure LVGL, how to create and manage graphical objects, how to handle user input and events, and how to add themes and styles. These guides can help users quickly get started with LVGL and develop their own applications.
- API documentation: The documentation provides a comprehensive list of LVGL's API interfaces and functions, along with their parameters and usage. Users can consult the API documentation to understand the specific functions and usage of individual functions and interfaces, enabling more advanced customization and extension.
- Example code: The documentation provides numerous example codes covering common application scenarios and functionalities. Users can leverage these example codes to accelerate development and quickly implement specific functionality requirements.

Using the LVGL online documentation can help users better understand and master the usage and techniques of LVGL, improving development efficiency. Users can gradually learn the contents of the documentation, starting from simple interface construction to complex application development, gradually mastering the various features and capabilities of LVGL. Additionally, the documentation provides examples and code snippets, making it easier for users to develop applications with rich interfaces and functionality.

Users can access the LVGL online documentation through a web browser and browse through the chapters and contents to find and learn relevant knowledge according to their needs. Additionally, the documentation provides a search function to quickly find specific information within the documentation. In summary, the LVGL online documentation is an important resource for users to understand and use the LVGL graphics library. It provides comprehensive and detailed guidance to help users quickly get started and develop better applications.

It is worth noting that while developing based on the documentation can complete most of the UI effects, the documentation may not be exhaustive. When there are omissions in the documentation, the code should be considered the most reliable source.

Github Repo

- Github LVGL

The LVGL GitHub repository is an important platform for developers to use and contribute to LVGL:

- Getting the latest version: The LVGL GitHub repository provides access to the latest LVGL versions and updates. Developers can stay up-to-date with the latest feature updates, bug fixes, and improvements, keeping their applications in sync with LVGL.
- Engaging in the community and contributing code: Through the GitHub repository, developers can actively participate in LVGL community discussions and exchanges, learning about other developers' issues and solutions. At the same time, developers can contribute their own code and improvements, making LVGL more robust and powerful.
- Submitting issues and bug reports: The GitHub repository offers a platform for issue and bug reporting, allowing developers to submit problems and bugs encountered during their use of LVGL. This helps the LVGL development team promptly discover and resolve issues, improving the stability and reliability of LVGL.
- Learning from examples and documentation: The GitHub repository also includes example code and documentation to help developers better understand and learn how to use LVGL. By browsing the repository's example code and documentation, developers can learn about the various features and capabilities of LVGL, enhancing their development skills.

Designer

- GUI Guider: Free
- Squareline: Squareline Studio, Paid

The LVGL Designer is a visual tool for designing and developing interfaces for the LVGL graphics library. It provides an intuitive and user-friendly interface that allows developers to quickly create and edit GUI interfaces using LVGL.

The LVGL Designer has the following features and functionalities:

- Visual interface design: The designer provides an intuitive visual interface where developers can create and edit GUI interfaces using mouse and simple drag-and-drop operations. It allows adding and adjusting various graphical objects, labels, buttons, text boxes, images, and more, while setting their size, position, style, and other attributes.
- Real-time preview and debugging: The designer supports real-time preview, allowing developers to see the appearance and behavior of the designed interface at any time. This helps developers quickly adjust and optimize the interface to achieve the desired effect.
- Event and interaction management: The designer enables developers to conveniently add and manage events and interaction behaviors. Developers can add click, scroll, drag, and other events to graphical objects and configure their response behaviors through simple configurations.
- Theme and style customization: The designer supports customization of themes and styles, allowing developers to easily select and apply different themes and styles to make the interface more personalized and visually appealing.
- Code export: The designer allows exporting the designed interface as LVGL code, providing the necessary initialization and configuration. This enables developers to directly use the exported code for LVGL application development, eliminating the need for manual code writing.

Using the LVGL Designer greatly accelerates the design and development process of GUI interfaces, especially for non-professional UI designers or developers. With simple drag-and-drop and configuration operations, developers can quickly

create attractive and interactive interfaces, improving development efficiency and user experience. Additionally, the designer provides a convenient way to export the designed interface as usable LVGL code, allowing developers to easily integrate it into their applications.

Forum

- LVGL Forum

The official LVGL forum is a developer community dedicated to discussing and sharing topics and resources related to the LVGL graphics library. It provides a platform for developers to exchange ideas, seek help, and share their experiences and projects.

Some features and functionalities of the LVGL forum include: - Questions and answers: Developers can ask questions about their LVGL usage on the forum and receive help and answers from other developers. This makes the forum a valuable knowledge base, providing experience and tips for problem-solving.

- Tutorials and examples: The forum contains many useful tutorials and example code, demonstrating how to use different features and functionalities of LVGL. These resources are helpful for novice developers to learn and master LVGL.
- Developer contributions and project showcases: Developers on the forum can share their projects and customized LVGL interfaces, as well as contributions that other developers can share, discuss, and reference.
- Updates and release announcements: The LVGL development team provides announcements and explanations about new version releases and updates on the forum. This allows developers to stay informed about the latest features and improvements.
- Community interaction: The forum provides a platform for community interaction, where developers can communicate, share, and establish connections, enhancing collaboration and development within the LVGL community.

The LVGL forum is an important resource for developers using LVGL to receive support, solve problems, learn, and share experiences.

Blog

- LVGL Blog

The official LVGL blog is a regularly updated platform that provides the latest information, tutorials, case studies, and developer insights about the LVGL graphics library. The LVGL development team and community members frequently publish various content related to LVGL on the blog, helping developers better understand and use LVGL.

The LVGL blog covers the following content: - Updates and new feature introductions: The blog publishes articles on the latest version of LVGL, highlighting new features, bug fixes, and performance improvements. This allows developers to stay up-to-date and leverage the latest LVGL capabilities.

- Tutorials and guides: The blog provides practical tutorials and guides on LVGL, covering various topics ranging from beginner to advanced. These tutorials often include example code and detailed explanations, helping developers master the usage of LVGL and best practices.
- Case studies and project showcases: The blog shares case studies and project showcases implemented with LVGL. These articles demonstrate how to use LVGL to build real-world applications and interfaces, providing developers with inspiration and experience from practical implementations.
- Technical deep dives and developer insights: The blog also covers in-depth analyses of LVGL and insights from developers. These articles may explore topics such as the internal workings of LVGL, performance optimization techniques, and excellent design practices, providing developers with a deeper understanding and food for thought.

The LVGL blog is an important resource for developers to understand and master LVGL. By reading the blog, developers can gain insights on the latest LVGL developments, learning materials, and technical insights, helping them utilize LVGL to build excellent graphical interfaces.

2.2.8 FAQ

- LVGL FAQ

HoneyGUI vs LVGL Picture Drawing Frame Rate

GRAM Screen (280x456) RAM Block Drawing

Background: RTL8772G, RGB565, uncompressed images, test for the performance of displaying a single image (HoneyGUI rectangle fill data is temporarily unavailable; LVGL has not adapted PPE hardware acceleration for image scaling yet).

Table 3: RAM Block Drawing

Test Case	HoneyGUI FPS (SW)	HoneyGUI FPS (PPE)	LVGL FPS (SW)	LVGL FPS (PPE)
Draw Image	73	74	70	73
Fill Rectangle	3	85	74	74
Rotate Image 45°	3	3	4	4
Scale Up 1.5x	3	31	3	25
Scale Down 0.5x	9	73	12	25

Table 4: RAM Block Drawing Test Data

Section	HoneyGUI FPS	LVGL FPS
10	70	45
20	73	73
30	74	73

PSRAM Full Frame Buffer Drawing (800x480)

Background: RTL8772G, RGB565, image size 315x316, uncompressed images, RGB screen, test for the performance of displaying a single image.

Table 5: PSRAM Full Frame Buffer Drawing

Test Case	HoneyGUI FPS (SW)	HoneyGUI FPS (PPE)	LVGL FPS (SW)	LVGL FPS (PPE)
Draw Image	76	76	17	25
Fill Rectangle	4	78	25	26
Rotate Image 45°	3	3	6	4
Scale Up 1.5x	2	23	3	13
Scale Down 0.5x	10	82	13	50

Analysis

Extra PSRAM is required for RGB screen as a cache buffer. LVGL uses PSRAM completely as its image cache buffer compared to HoneyGUI which combines RAM and PSRAM. LVGL performs worse overall.

HoneyGUI vs LVGL RAM Consumption

Table 6: GRAM Screen (280x456) Dynamic RAM Consumption

Test Case	HoneyGUI (Bytes)	LVGL Widget Consumption (Bytes)
Draw Image	156	176
Fill Rectangle	64	200
Rotate Image 45°	156	208
Scale Up 1.5x	156	208
Scale Down 0.5x	156	176

Table 7: GRAM Screen (280x456) Static RAM Consumption

Test Case	HoneyGUI (Bytes)	LVGL Widget Consumption (Bytes)
Draw Image	41892(40KB)	55300(54KB)
Fill Rectangle	41892(40KB)	55300(54KB)
Rotate Image 45°	41892(40KB)	55300(54KB)
Scale Up 1.5x	41892(40KB)	55300(54KB)
Scale Down 0.5x	41892(40KB)	55300(54KB)

Conclusion

- **Applicable Scenarios**: For large screen sizes (e.g. 800x480) and full-frame drawing, HoneyGUI is recommended. For frequent partial screen refresh projects, LVGL is recommended. For block drawing when RAM resources are tight, HoneyGUI is recommended, with section recommended parameters set to 10.
- **Rotation, Scaling**: LVGL performs faster in 2D rendering using a 2x2 matrix compared to HoneyGUI's 3x3 matrix, which handles more data for 2D rendering. For 2.5D or pseudo-3D effects, HoneyGUI will perform better.
- In practical projects, select the suitable framework based on specific frame rate requirements, system resources, and other functional needs. Conduct specific performance testing and evaluation if possible.

This analysis provides valuable insights for selecting the appropriate display framework and assists decision-makers in making the best choice based on actual requirements.

2.3 Use ARM-2D Design An Application

Arm-2D is an open-source project for 2.5D image processing on Cortex-M processors.

- Initial Targets: IoT terminal devices, white appliances, handheld devices, and wearable devices, especially for resource-constrained and low-power-demand devices.
- Initial Focus: Graphical User Interface (GUI) development.

2.3.1 ARM-2D Introduction

- ARM-2D

2.4 Use RVD Tool Design An Application

2.4.1 Overview

RTKIOT Visual Designer is a tool to create graphical interface designs for Realtek series ICs; its currently supported ICs are shown in the table below.

Table 8: Supported ICs

No	Supported ICs
1	RTL8762D
2	RTL8762G
3	RTL8763E
4	RTL8772G
5	TBD

RTKIOT Visual Designer supports:

- Drag the widgets from the toolbox and drop them in the Design View.
- Drag and drop the widget to change its position in the Design View, or modify the position and appearance of the widget via the Property View.
- Export the user-designed GUI project to `.bin`, and the `.bin` can be programmed into the IC to display the graphical interface.
- Simulate the GUI project on a PC.

This document mainly consists of:

- *Function Panels*
- *Resource Management*
- *Menu Bar*
- *Quick Start to Tutorials*
- *GUI Demo Project*

To simplify the document, *Tool* is used below to refer to the *RTKIOT Visual Designer*.

2.4.2 Function Panels

Toolkit/Widgets

- Non-containerized widget
  - Can be used as the parent of other widgets.
  - There is a coordinate-following relationship between parent and child widgets.
  - Visible when the child widget is out of range of the parent widget.

- Container widget
  - Can be used as the parent of other widgets.
  - There is a coordinate-following relationship between parent and child widgets.
  - Visible when the child widget is out of range of the parent widget.
  - Can drag and drop a widget from the toolbox into the container widget.

This section lists the properties supported by the widget in tables and marks with **Y** or **N** to indicate whether the IC supports them or not.

Non-containerized Widget

Text

Used only for text display and does not support user input. The properties are shown in the table below.

Table 9: Text Widget Properties

Property	Description	8762D	8762G	TBDG
Name	Widget name.	Y	Y	Y
Size (Height)	Widget height.	Y	Y	Y
Size (Width)	Widget width.	Y	Y	Y
X	Horizontal coordinate relative to the parent widget.	Y	Y	Y
Y	Vertical coordinate relative to the parent widget.	Y	Y	Y
Text	Display text.	Y	Y	Y
Display Mode	Long text (text content beyond the widget's range) display mode with the following supported types. truncate: Truncated display mode; verticalscroll: Vertical scrolling display mode; horizontalscroll: Horizontal scrolling display mode.	Y	Y	Y
Font	Font setting, please refer to *Font Convert Setting*.	Y	Y	Y
Font Color (RGBA)	Font color setting, use RGBA.	Y	Y	Y

Button

Clickable widget with text and background image. The properties are shown in the table below.

Table 10: Button Widget Properties

Property	Description	8762[	8762(	TBD
Name	Widget name.	Y	Y	Y
Size (Height)	Widget height.	Y	Y	Y
Size (Width)	Widget width.	Y	Y	Y
X	Horizontal coordinate relative to the parent widget.	Y	Y	Y
Y	Vertical coordinate relative to the parent widget.	Y	Y	Y
Text	Displayed text.	Y	Y	Y
Text X	Horizontal coordinate relative to the Button widget.	Y	Y	Y
Text Y	Vertical coordinate relative to the Button widget.	Y	Y	Y
Display Mode	Horizontal or Vertical display.	Y	Y	Y
Font	Font setting, please refer to *Font Convert Setting*.	Y	Y	Y
Text Color (RGB)	Text color setting, use RGB.	Y	Y	Y
Transition	Image transition mode with the following options: normal: No effect fade: Fade-in/out scale: Scaling fadeScale: Fade-in/out and scaling Note: Set the transition mode is effective only if set the default and highlight background image, otherwise all normal.	N	Y	Y
BG Image (Default)	Default background image.	Y	Y	Y
BG Image (Highlight)	Selected/Highlight background image.	Y	Y	Y
BG Image Rotation Angle	Background image rotation angle, range: 0~360 degree.	Y	Y	Y

Image

Table 11: Image Widget Properties

Property	Description	8762D/	8762G/	TBD
Name	Widget name.	Y	Y	Y
Size (Height)	Widget height.	Y	Y	Y
Size (Width)	Widget width.	Y	Y	Y
X	Horizontal coordinate relative to the parent widget.	Y	Y	Y
Y	Vertical coordinate relative to the parent widget.	Y	Y	Y
Image	Image Path Note: The image must be pre-imported into the project. Please refer to *Image Resource Management* for details.	Y	Y	Y
Image Rotation Angle	Image rotation angle.	Y	Y	Y
Image Scale X	Image horizontal scaling degree, is a multiplier/percentage. For example, set scale x 0.5 means that the actual display width of the image is half of the original image width.	Y	Y	Y
Image Scale Y	Image vertical scaling degree, is a multiplier/percentage.	Y	Y	Y

Widget that can set image. The properties are shown in the table below.

Note:

1. When exporting, the tool will convert the imported images. And the image conversion parameters can be set in *Menu Bar ▸ Setting ▸ Image Convert Setting*, please refer to *Image Convert Setting* for details;
2. If the size of the imported image does not match the size of the widget, the tool doesn't scale or crop the image.

SeekBar

Sliding widget that can respond to user swipe gesture with the widget and change the progress value. The properties are shown in the table below.

Fig. 2: SeekBar

Table 12: SeekBar Widget Properties

Property	Description	8762D/	8762G/	TBD3
Name	Widget name.	Y	Y	Y
Size (Height)	Widget height.	Y	Y	Y
Size (Width)	Widget width.	Y	Y	Y
X	Horizontal coordinate relative to the parent widget.	Y	Y	Y
Y	Vertical coordinate relative to the parent widget.	Y	Y	Y
Color(Highli (RGBA)	Background color of partially completed part of the progress bar.	N	Y	N
Color (RGBA)	Background color of the whole progress bar.	N	Y	N
Orienta-tion	Widget display orientation and gesture response orientation with the following types: vertical/V: Vertical orientation arc: Direction of a curve horizontal/H: Horizontal orientation	Y	Y	Y

Image SeekBar

Sliding widget with multi-images as background, and switch to different images as the user swipes. The properties are shown in the table below.

Table 13: Image SeekBar Widget Properties

Property	Description	8762D/8763E	8762G/8772G	TBD
Name	Widget name.	Y	Y	Y
Size (Height)	Widget height.	Y	Y	Y
Size (Width)	Widget width.	Y	Y	Y
X	Horizontal coordinate relative to the parent widget.	Y	Y	Y
Y	Vertical coordinate relative to the parent widget.	Y	Y	Y
Degree (Start)	Start degree (Invalid if orientation is arc).	Y	Y	Y
Degree (End)	End degree (Invalid if orientation is arc).	Y	Y	Y
Image Directory	Folder that contains only the images to be displayed on this widget. Notes: 1. Please sort the images by name; 2. When the user swipes on the widget, the widget will switch the background image according to the current progress.	Y	Y	Y
Central X	Horizontal coordinate of the center of the arc relative to the parent widget.	Y	Y	Y
Central Y	Vertical coordinate of the center of the arc relative to the parent widget.	Y	Y	Y
Orientation	Widget display orientation and gesture response orientation with the following types: vertical/V: Vertical orientation arc: Direction of a curve horizontal/H: Horizontal orientation	Y	Y	Y

Switch

Switch widget with **Checked** and **Unchecked** states. The properties are shown in the table below.

Table 14: Switch Widget Properties

Property	Description	8762D/8763E	8762G/8772G	TBD
Name	Widget name.	Y	Y	Y
Size (Height)	Widget height.	Y	Y	Y
Size (Width)	Widget width.	Y	Y	Y
X	Horizontal coordinate relative to the parent widget.	Y	Y	Y
Y	Vertical coordinate relative to the parent widget.	Y	Y	Y
BG Image (Checked)	Checked state background image.	Y	Y	Y
BG Image (Default)	Unchecked state background image.	Y	Y	Y

Arc

Arc widget, no gesture support yet. The properties are shown in the table below.

Table 15: Arc Widget Properties

Property	Description	8762D/8763E	8762G/8772G	TBD
Name	Widget name.	Y	Y	N
Size (Height)	Widget height.	Y	Y	N
Size (Width)	Widget width.	Y	Y	N
X	Horizontal coordinate relative to the parent widget.	Y	Y	N
Y	Vertical coordinate relative to the parent control.	Y	Y	N
Central X	Horizontal coordinate of the center of the arc relative to the parent widget.	N	Y	N
Central Y	Vertical coordinate of the center of the arc relative to the parent widget.	N	Y	N
BG Color	Arc background color.	N	Y	N
Cap Mode	Arc cap mode, the following options are supported: round/butt/squa	N	Y	N
Degree (End)	End degree of arc.	N	Y	N
Degree (Start)	Start degree of arc.	N	Y	N
Radius	Radius of arc.	N	Y	N
Stroke Width	Width of arc stroke.	N	Y	N

Container Widget

Screen

Screen widget, corresponding to the physical screen, is the root widget of a GUI project. The properties are shown in the table below.

Table 16: Screen Properties

Property	Description	8762D/8763E	8762G/8772G	TBD
Name	Widget name.	Y	Y	Y
Size (Height)	Widget height.	Y	Y	Y
Size (Width)	Widget width.	Y	Y	Y
X	Horizontal coordinate, always 0.	Y	Y	Y
Y	Vertical coordinate, always 0.	Y	Y	Y

Note: Only 'Name' property can be modified.

TabView and Tab

With the Tab widget as a child widget, it supports up/down/left/right swiping to switch among Tabs. The properties of TabView and Tab are shown in the table below.

Fig. 3: TabView and Tabs

Table 17: TabView Properties

Property	Description	8762D/8	8762G/8	TBD
Name	Widget name.	Y	Y	Y
Size (Height)	Widget height.	Y	Y	Y
Size (Width)	Widget width.	Y	Y	Y
X	Horizontal coordinate relative to the parent widget, always 0.	Y	Y	Y
Y	Vertical coordinate relative to the parent widget, always 0.	Y	Y	Y
Transition	Tab transition mode with the following supported types: normal: No effect fade: Fade-in/out scale: Scaling fadeScale: Fade-in/out and scaling	N	Y	Y

Table 18: Tab Properties

Property	Description	8762D/8763E	8762G/8772G	TBD
Name	Widget name.	Y	Y	Y
Size (Height)	Widget height.	Y	Y	Y
Size (Width)	Widget width.	Y	Y	Y
X	Horizontal coordinate relative to TabView widget, always 0.	Y	Y	Y
Y	Vertical coordinate relative to TabView widget, always 0.	Y	Y	Y
Index(X-Axis)	Horizontal index of Tabs in TabView.	Y	Y	Y
Index(Y-Axis)	Vertical index of Tabs in TabView.	Y	Y	Y

Note:

1. TabView width and height cannot be modified, defaulting to the Screen's width and height;
2. TabView horizontal and vertical coordinates cannot be modified, always being 0;
3. TabView can only be used as a child of the Screen widget;
4. TabView's child widgets can only be Tabs;

5. Tab's width and height cannot be modified, defaulting to TabView's width and height;
6. Tab's horizontal and vertical coordinates cannot be modified and are always 0.

Page

Container widget with scrollable content.

Table 19: Page Properties

Property	Description	8762D/8763E	8762G/8772G	TBD
Name	Widget name.	Y	Y	Y
Size (Height)	Widget height.	Y	Y	Y
Size (Width)	Widget width.	Y	Y	Y
X	Horizontal coordinate relative to the parent widget.	Y	Y	Y
Y	Vertical coordinate relative to the parent widget.	Y	Y	Y

Note:

1. Page only supports vertical scrolling;
2. The width and height of the Page widget only define the area of the interface that can respond to a swipe gesture. Whether scrolling is allowed depends on whether or not the child widget added to it is outside the scope of the screen.

Win

Within the area defined by Win width and height, it can respond to various gestures, including click, long click, press, press release, and swipe. The properties are shown in the table below.

Table 20: Win Properties

Property	Description	8762D/8763E	8762G/8772G	TBD
Name	Widget name.	Y	Y	Y
Size (Height)	Widget height.	Y	Y	Y
Size (Width)	Widget width.	Y	Y	Y
X	Horizontal coordinate relative to the parent widget.	Y	Y	Y
Y	Vertical coordinate relative to the parent widget.	Y	Y	Y
Hidden	Indicates whether Win and its child widget need to be hidden.	Y	Y	Y

Design View/Canvas

Users can drag and drop widgets from the Toolbox panel into the Design View, adjust the widgets' layout, and set properties to design a graphical interface that can be rendered in the Realtek ICs.

Fig. 4: Design View

TabView - Create/Delete/Insert Tab

Drag and drop the TabView widget from the Toolbox into the Design View, then a TabView that contains only a home tab (coordinates (0,0)) is created, as shown in the figure below.

Fig. 5: Create TabView

Create Tab

New tabs can be created by clicking the buttons around the Design View.

Note:

1. If idx is 0, the up and down button is enabled;
2. If idy is 0, the left and right button is enabled.

Delete Tab

Select the tab to be deleted, click *Edit* ▸ *Delete* on the menu bar or press the `Delete` key on the keyboard. Then double-check if the deletion is intended.

Fig. 6: Delete Tab Double-Check

Insert Tab

Currently, tab insertion is only supported by modifying the coordinates of an existing tab and creating a new one.

For example, if a tab needs to be inserted between tabs with coordinates (1, 0) and (2, 0), the steps are as follows.

1. Increase the idx of Tab (2, 0) and the tabs to its right by 1, as shown in the figure below;
2. Switch to Tab (1, 0) and click to create the new Tab (2, 0).

Fig. 7: Tab Insertion Position

Fig. 8: Modify Existing Tab Index X and Y

Fig. 9: Insert Tab

TabView Overview Window

Please click to show the *TabView Overview Window*.

Note:

1. The highlighted Tab in the Overview Chart indicates the Tab that is currently being edited in Design View;
2. The Overview Chart labels each Tab with its coordinates. When simulated or rendered in ICs, the Tab with coordinates (0,0) is displayed on the Home page, and users can swipe up/down/left/right to display other Tabs.

Fig. 10: TabView Overview Chart

Fig. 11: TabView Overview Chart

Zoom of Design View

There are 3 ways to zoom in the Design View.

1. Press the `Ctrl` key and wheel mouse;
2. Click the - and + buttons;
3. Drag the slider bar.

Fig. 12: Zoom of Design View

Property View

Selecting a widget in the Widget Tree or Design View exposes all of the widget's property values, which users can modify as needed.

Fig. 13: Property View

Widget Tree

The Widget Tree is used to present to the users the parent/child/sibling relationship of the currently laid out widgets. And we have the following convention here.

1. The child widget layer is on top of the parent widget layer, i.e., when the parent and child widget overlap, the child widget will cover the parent widget;
2. The layer of sibling widgets is related to the order in which the widgets are added, with widgets added first at the bottom and widgets added later at the top.

The figure shows all the child widgets of the Home tab and Lamp tab, where the Home tab has only one Image child widget for setting the background, and the Lamp tab contains an Image widget and several Switch widgets.

Fig. 14: Home Tab

Fig. 15: Lamp Tab

Widget Tree supports the following operations.

1. Select widget: If a widget is selected on the Widget Tree, the corresponding widget in the Design View focuses and its properties are shown on Property View;
2. Modify the parent-child relationship: Select a widget on the Widget Tree (except Tab/TabView/Screen) and drag-and-drop it on the target widget item. Then the widget will be a child widget of the target widget;
3. Modify widget layers: Select a widget on the Widget Tree (except Tab/TabView/Screen) and drag-and-drop it to the upper or lower edge of the target widget item. Then on the Design View, the widget will be placed over or under the target widget;
4. Lock widgets: Click the button and lock the widget/widgets.
    1. If the lock button of the screen is clicked, all the screen's child widgets will be locked, and the user could not drag or resize the widgets on Design View;
    2. If the lock button of the Tab is clicked, all the tab's child widgets will be locked, and the user could not drag or resize the widgets on Design View.

Fig. 16: Un-Locked

Fig. 17: Locked

2.4.3 Resource Management

Only pre-imported image and font files can be referenced by the GUI project. This chapter focuses on how to manage image and font resources. The image and font explorer is located directly below the design view, as shown in the figure below.

Fig. 18: Image Resource Management

Font Resource Management

Image Resource Management

Click to bring up the Image Management view.

Image Resource Management Window

Add Images

Images can be added to the GUI project by following the process below.

1. Click to create a new image folder and enter the folder name. The created folder is located in the `Resource\image` folder under the GUI project directory.

Fig. 19: Create Image Folder

2. Select the created image folder and click to select images (multiple selections are possible) to add them to the folder. As shown in the figure below, the images are copied to the `Resource\image\home` folder after the addition is completed.

Fig. 20: Select Image Folder

Fig. 21: Select Images

Fig. 22: Add Image(s)

Remove Images/Image Folder

Select the image or image folder to be removed and click .

Rename Image Folder

Select the image folder, double-click, and enter a new name.

Preview Images

Select the image folder and all images in this folder will be displayed in the right area.

Fig. 23: Preview Images

Refresh

If the user locally operates the image resources, not via Tool, click to refresh.

Note: Not recommended.

Font Resource Management

Add Third-Party Font

If a third-party font (`.ttf`) is needed, click to import the resource first; otherwise, the locally installed font will be used.

Fig. 24: Font Management

Remove Third-Party Font

Select the font to be removed and click .

2.4.4 Menu Bar

File

Start Page

To close the current project and open an existing project or create a new project, open the Start Page by clicking *File* ▸ *Start Page*. Click *Open Project* or select a `.rtkprj` and double-click to open the existing project, or click *Create Project* to create a new project. Please refer to *How to Create Project* and *How to Open Project*.

Fig. 25: Start Page

Save

Save all the UI changes of the project, the shortcut is `Ctrl + S` .

Exit Save

A prompt window will pop up when closing the project, as shown below. Please click *OK* to save, or the changes will be abandoned.

Fig. 26: Close and Save Project

Edit

Copy/Paste

1. Click *Edit* ▸ *Copy* to copy the selected widget, the shortcut is `Ctrl + C`.
2. Click *Edit* ▸ *Paste* to create a copy of the selected widget on the Design View, the shortcut is `Ctrl + V`.

Delete

Click *Edit* ▸ *Delete* to delete the selected widget, or press the `Delete` key on the keyboard.

Undo/Redo

Undo: Undo the change made to the widget, the shortcut is `Ctrl + Z`. Redo: Do the change to the widget again, the shortcut is `Ctrl + Y`.

Convert Project

The Convert Project window is used to convert the IC type and screen size/resolution for the current project.

Fig. 27: Convert Project

Project Name Modification

The Project Name window is used to modify the current project name. Please enter the new name in the input box.

Fig. 28: Project Name

Setting

Image Convert Setting

The images must be converted to be displayed correctly on the IC, so users need to set the correct convert parameters. All the optional parameters are shown in the figure below.

Fig. 29: Image Convert

The parameters are described as follows.

Scan Mode

The available options are shown in the table.

Table 21: Scan Mode Options

Scan Mode	Description
Horizontal	Horizontal scan.
Vertical	Vertical scan.

Color Space

Color space of Image, the available options are shown in the table below.

Table 22: Color Space Options

Color Space	Description
RGB565	16 bit RGB mode Bit 4:0 represents blue; Bit 10:5 represents green; Bit 15:11 represents red.
RTKARGB	16 bit ARGB mode Bit 4:0 represents blue; Bit 9:5 represents green; Bit 14:10 represents red; Bit 15 represents alpha.
RTKR-GAB	16 bit RGAB mode Bit 4:0 represents blue; Bit 5 represents alpha; Bit 10:6 represents green; Bit 15:11 represents red.
RGB	24 bit RGB mode Bit 7:0 represents blue; Bit 15:8 represents green; Bit 23:16 represents red.
RGBA	32 bit RGBA mode Bit 7:0 represents blue; Bit 15:8 represents green; Bit 23:16 represents red; Bit 31:24 represents alpha.
BINARY	2-value (0 or 1) image.

Compress

If checked *Compress*, please set the compression parameter as needed. The optional Compress Mode is as follows:

1. RLE

Run-Length Encoding, a lossless compression algorithm.

If selecting RLE as the Compress Mode, RLE Level and RLE Run Length parameters are mandatory to set.

Fig. 30: RLE Level - Level 1

Fig. 31: RLE Level - Level 2

Table 23: RLE Level

RLE Level	Description
Level 1	1-level compress.
Level 2	2-level compress, secondary compress based on the 1-level compress.

Table 24: RLE Run Length

RLE Run Length	Description
Byte_1	1 byte, Maximum 255.
Byte_2	2 bytes, Maximum 255.

Note: RLE Run Length: Maximum length of duplicate characters allowed per stroke (Run) during compression.

2. FastLz

   A dictionary-and-sliding-window based lossless compression algorithm for compressing data with a large number of repetitive values.

3. YUV_Sample_Blur

   A lossy compression algorithm combining YUV sampling and blurring.

   YUV Sample: Keep the luminance information of the image and only sample the chrominance information.

   Blur: Discard the lower bit of each byte after YUV sampling to achieve the purpose of data compression.

Table 25: YUV Sample Mode

YUV Sample Mode	Description
YUV444	4 pixel data are sampled to 4 Y, 4 U and 4 V, i.e., each Y corresponds to a set of UV components, fully preserving the YUV data.
YUV422	Every 4 pixel data are sampled to 4 Y, 2 U and 2 V, i.e., every 2 Y corresponds to a set of UV components, data size is 75% of the original.
YUV411	Every 4 pixel data are sampled to 4 Y, 1 U and 1 V, i.e., every 4 Y corresponds to a set of UV components, data size is 50% of the original.
YUV422	Y - luminance; V - chrominance.

Table 26: Blur Mode

Blur Mode	Description
Bit0	Saving without discarding lower bit.
Bit1	Each byte discards bit0 (preserve [bit7:bit1]).
Bit2	Each byte discards [bit1:bit0] (preserve [bit7:bit2]).
Bit4	Each byte discards [bit3:bit0] (preserve [bit7:bit4]).

4. YUV_Sample_Blur+FastLz

The algorithm combines YUV_Sample_Blur and FastLz.

Font Convert Setting

Include Bitmap Fonts and Vector Fonts. Fonts supported by Realtek series ICs are shown in the table below.

Note: A Font Convert Setting should be created, otherwise selecting a font for the text-type widget in the Property View is not possible.

Table 27: Supported Fonts Type

Font	8762D/8763E	8762G/8772G	TBD
Vector	N	N	Y
Bitmap	Y	Y	Y

To use Bitmap Fonts, set the following parameters.

Fig. 32: Convert Settings of Bitmap Fonts

The following table lists the description of each parameter.

Table 28: Font Convert Parameters

Parameters	Description
Font Setting Name	User-defined font setting name. Please make sure that you create a different font setting name each time.
Font Size	Font size.
Bold	Bold or not.
Italic	Italic or not.
Render Mode	Bit number used to represent a pixel in the converted `.bin` file.
Scan Mode	There are two ways to scan when saving `.bin`. H: Horizontal scanning V: Vertical scanning
Index Method	Index method of the converted .bin's re-indexing area.
Code Page	Support multiple code pages.
Text Type	The types are as follows. Range: If the text's Unicode range can be pre-determined, please select this type and enter the range in the Range TextBox. Multiple ranges are supported, please set each range on a separate line. Note: Only the characters within the ranges will be converted and saved to `.bin` file, which can save storage space. Random: If the text's Unicode range cannot be pre-determined, please select this type. Note: All characters of the Font will be converted and saved to `.bin` file.

Vector Font parameters are shown in the figure below.

Fig. 33: Vector Font Parameters

Export

If you have finished designing the GUI project and want to program it to the IC, please click *Export*, then the Tool performs the following actions:

1. Image convert
2. Font convert
3. Pack the `.xml`, `.js`, images and fonts into the output `.bin`.

When the above actions are done, a message box pops up.

Fig. 34: Output .bin

The `.bin` can be programmed into your IC.

Simulate

Simulate the project on UI.

Note: When simulating the project for the first time, please click *Export* before clicking *Simulate*. Then, there is no need to click *Export* again if no image or font setting is modified.

Fig. 35: Running Simulator

2.4.5 Quick Start to Tutorials

How to Create Project

Fig. 36: Start Page

Double click and run `RVisualDesigner.exe`, and then configure the project step by step (1~4) and click *Create Project* (5). After creation, the GUI design window pops up. The left side is the component area, the center is the design area, and the right side is the widget property setting area.

Fig. 37: GUI Design

Note: The newly created project file is located in the project folder under the Solution Folder. There is an example as shown in the figure below.

Fig. 38: Project Folder

After dragging and dropping a widget on Design View, and clicking *File* ▸ *Save* or pressing `Ctrl + S`, the `.rtkui` file will be created.

Fig. 39: .rtkui File

How to Write Javascript Code

After the project is created, the `xxx.js` file is created. The `xxx.js` file is empty, please code here to implement the widgets' event callback.

How to Open Project

Fig. 40: Open Project

There are two ways to open a project.

1. Click *Open Project* and select a `.rtkprj` file.

Fig. 41: Open Project via Selecting .rtkprj

2. Select a `.rtkprj` in the Recent Project area.

If the project is listed in the Recent Project area, a message window pops up.

Fig. 42: Message Box

How to Open/Close Project

Click *File* ▸ *Start Page* on Menu Bar.

How to Export/Pack Project

Fig. 43: Export

Click *Export* on Menu Bar. The output is shown in the figure below.

Fig. 44: Export OK

How to Simulate

Fig. 45: Simulate

Click on the *Simulate* button in the menu bar.

2.4.6 GUI Demo Project

There is a Demo in `RVisualDesigner-vx.x.x.x.zip`.

The folder - 454x454 contains a project with resolution 454*454.

The folder - 480x480 contains a project with resolution 480*480.

Fig. 46: Demo

Please follow the steps to demo the project.

1. Open the project according to the screen size/resolution of your IC;
2. Check the IC type by clicking *Edit ▸ Convert Project* on the Menu Bar. Please refer to *Convert Project* for details. If the current IC type of the project does not match your IC, please select the target IC type, enter the target resolution, and click *Convert*.

Fig. 47: Convert Project

3. Click *Export* on the Menu Bar and wait until the export ok/fail message box pops up.

Fig. 48: Output .bin

Program the output `.bin` into your IC.

2.4.7 JavaScript Syntax

Win

- This is a container widget.
- Operations on the window widget will affect the widgets nested in the container.
- Hiding the window will hide the nested widgets.
- When the window makes graphic transformations, such as panning and scaling, the nested widgets will make consistent transformations.
- This widget can monitor multiple gestures.

Hide A Window

- This `win` variable is assigned the `win` tag `'heat_win'`'s handle.
- The variable `hid` is assigned the handle of the `hidden` attribute of the `win` tag.
- The value of the `hidden` attribute is set to `'hidden'` to achieve hiding.

```
win.getElementById('heat_win') //win will become a handle for heat_win
hid = win.getAttribute("hidden") //get attribute handle hid
console.log(hid)
if (!hid) {
```

(continues on next page)

(continued from previous page)

```
    win.setAttribute("hidden", "hidden");
}
```

Listen to Gestures

- The `win.onPress` function enables the win widget to monitor the event of the finger touching the screen. If the finger touches the screen within the area of the window, the parameter function will be executed.
- The `win.onRelease` function enables the win widget to monitor the event of the finger leaving the screen.
- This `winNromalOnPressFunc` function will be executed when the finger touches the screen.
- This `winNromalOnReleaseFunc` function will be executed when the finger leaves the screen.

```
win.getElementById('tab7Win')
function winNromalOnPressFunc(params) {
    console.log('winNromalOnPressFunc')
}
win.onPress(winNromalOnPressFunc)

function winNromalOnReleaseFunc(params) {
    console.log('winNromalOnReleaseFunc')
}
win.onRelease(winNromalOnReleaseFunc)
```

Swap Windows

- The implementation logic is that clicking the current window will hide the current window and display another window.
- Click to swap windows between `'cool_win'` and `'heat_win'`.
- The `win.onClick` function enables the win widget to monitor the event of the finger clicking the screen.
- This function `win.removeAttribute` is used to remove an attribute of the win tag. When the `hidden` attribute is removed, the widget corresponding to the `win` tag will be displayed.
- On a touch device, a click event is typically triggered when a user touches an element and then lifts their finger in a short time within the win area.

```
win.getElementById('cool_win')
function hideCool(params) {
    console.log('hideCool')
    win.getElementById('cool_win')
    win.setAttribute("hidden", "hidden");
    win.getElementById('heat_win')
    win.removeAttribute("hidden")
}
win.onClick(hideCool)

win.getElementById('heat_win')
function hideHeat(params) {
    console.log('hideHeat')
    win.getElementById('heat_win')
    win.setAttribute("hidden", "hidden");
```

(continues on next page)

(continued from previous page)

```
    win.getElementById('cool_win')
    win.removeAttribute("hidden")
}
win.onClick(hideHeat)
```

API

```
getElementById : function (win_name : string) {}
onClick : function (callback_func) {}
onRight : function (callback_func) {}
onLeft : function (callback_func) {}
onUp : function (callback_func) {}
onDown : function (callback_func) {}
onPress : function (callback_func) {}
onRelease : function (callback_func) {}
onHold : function (callback_func) {}
getAttribute : function(attributeName : string) {},   //return attribute value //
↪support "hidden"
removeAttribute : function (attribute : string) {} //support "hidden"
setAttribute :function(attributeName : string, value : any) {},  //support "hidden"
```

Button

Monitor Button Press Event

- Can be used to develop button press highlight effects or buttons that require quick response.
- Listen to press gesture, the function `iconNromalOnPressFunc` will trigger when finger touches screen within the area of the button.

```
icon.getElementById('iconNormal')

function iconNromalOnPressFunc(params) {
    console.log('iconNromalOnPressFunc')
}
icon.onPress(iconNromalOnPressFunc)
```

API

```
getElementById : function (win_name : string) {},
onClick : function (callback_func) {},
onPress : function (callback_func) {},
onRelease : function (callback_func) {},
onHold : function (callback_func) {},
getChildElementByTag : function (tag : string) {},
write : function (text : string) {},
```

Text

Change Text Content

- Using `textbox.write` function.

```
textbox.getElementById('tab10text1')
textbox.write('progress:'+seekbar.progress())
```

API

```
getElementById : function (win_name : string) {},
write : function (text : string) {},
setPosition : function (position : object) {}, //var position={x:0,y:0}
```

Seekbar

Display Current Progress

- Drag the progress bar and then the text shows the current progress.
- Function `seekbar.progress` can read and write the progress.
- Function `seekbar.onPressing` will listen for events where your finger is kept pressed on the screen. This parameter function will be executed in each frame, while the finger is in contact with the screen.

```
seekbar.getElementById('tab10Seek1')
function seekbarOnPress(params) {
    console.log('seekbarOnPress')
}
seekbar.onPress(seekbarOnPress)
function seekbarOnrelease(params) {
    console.log('seekbarOnrelease')
}
seekbar.onRelease(seekbarOnrelease)
function seekbarOnPressing(params) {
    console.log('seekbarOnPressing')
    textbox.getElementById('tab10text1')
    textbox.write('progress:'+seekbar.progress())
}
seekbar.onPressing(seekbarOnPressing)
```

A Seekbar Animation That Increases From 0 to 100%

- The seekbar will display an animation that continuously progresses from start to finish and then loops back to the start, creating a perpetually moving progress bar.
- This function `seekbar.setAnimate` sets the frame animation of the seekbar, and the parameters passed are the frame animation callback and animation duration properties.
- Define an object `curtainAnimateTiming` to specify the timing properties for an animation. `duration` sets the duration of one cycle of the animation in milliseconds. `iterations` is the number of times the animation should repeat, and -1 indicates the animation should repeat indefinitely.

```
var curtainAnimateTiming = {
    duration: 2000,      // The duration of the animation in milliseconds (2000ms = 2 seconds)
    iterations: -1,      // The number of times the animation should repeat
                         // -1 indicates the animation should repeat indefinitely
};
var curtain_open = 0;
seekbar.getElementById('curtain_bar')
function curtainFrame(params) {
    animate= seekbar.animateProgress()
    seekbar.setAttribute("progress", animate)
}
seekbar.setAnimate(curtainFrame, curtainAnimateTiming)
seekbar.palyAnimate()
```

API

```
getElementById : function (win_name : string) {},
progress : function (progressToSet : number){},//get or set progress//return progress
onPress : function (callback_func) {}, //gesture press
onPressing : function (callback_func) {},//gesture pressing
onRelease : function (callback_func) {},//gesture release
setAnimate : function (frameCallback : function, config : object) {},// frameCallback function will be executed once every frame // var curtainAnimateTiming = {duration: 2000,iterations:1,}
setAttribute :function(attributeName : string, value : any) {},  //support "hidden"
getAttribute : function(attributeName : string) {},   //return attribute value // support "hidden"
palyAnimate : function () {}, //Start animation
```

Switch

Listen to 2 Gestures

- The switch widget has two events, namely, being triggered by being turned on and being triggered by being turned off.
- This function `sw.onOn` is used to register the turned on event.
- This function `sw.onOff` is used to register the turned off event.

```
sw.getElementById('tab8Switch')
function swOnOnFunc(params) {
    console.log('swOnOnFunc')
}
sw.onOn(swOnOnFunc)
function swOnOffFunc(params) {
    console.log('swOnOffFunc')
}
sw.onOff(swOnOffFunc)
sw.turnOn();
```

Turn on A Led (P1_1)

```
var P1_1 = 9
var LED1 = new Gpio(P1_1, 'out');
function led1OnFunc(params) {
    console.log('led1OnFunc')
    LED1.writeSync(0)
}
sw.getElementById('living_switch')
sw.turnOn()
```

- This is the `writeSync`'s control gpio led implementation for RTL87X2G.
- First get gpio value and direction value, then use specify driver api to operate led.
- Refer to onoff npm package usage for more information.

```
DECLARE_HANDLER(writeSync)
{
    gui_log("enter writeSync:%d\n", args[0]);
    if (args_cnt >= 1 && jerry_value_is_number(args[0]))
    {

        int write_value = jerry_get_number_value(args[0]);
        int gpio = -1;
        jerry_value_t v1;
        jerry_value_t v2;
        v1 = js_get_property(this_value, "gpio");
        v2 = js_get_property(this_value, "direction");
        gpio = jerry_get_number_value(v1);
        jerry_release_value(v1);
        char *direction = js_value_to_string(v2);
        jerry_release_value(v2);
        int mode = 0;
#ifdef RTL8762G

        if (!strcmp(direction, "out"))
        {
            mode = PIN_MODE_OUTPUT;
        }
        else if (!strcmp(direction, "in"))
        {
            mode = PIN_MODE_INPUT;
        }
        if (gpio >= 0)
        {
            gui_log("gpio%d, %d, %d", gpio, mode, write_value);
            drv_pin_mode(gpio, mode);
            drv_pin_write(gpio, write_value);
```

API

```
getElementById : function (win_name : string) {},
onOn : function (func) {},
onOff : function (func) {},
onPress : function (func) {},
turnOn : function (func) {},//turn on the switch
turnOff : function (func) {},//turn off the switch
```

Image

API

```
getElementById : function (widget_name : string) {},
rotation : function (degree:number, centerX:number, centerY:number) {},
scale : function (scaleRateX:number, scaleRateY:number) {},
setMode : function (modeIndex:number) {},
```

App

API

```
open : function (appXML : string) {},
close : function () {},
```

Progressbar

API

```
getElementById : function (widget_name : string) {},
progress : function (progressToSet : number):{},//get or set progress//return progress
```

Tab

API

```
getElementById : function (widget_name : string) {},
jump : function (tabIndex : number) {}, //jump to horizontal tab
OnChange : function (func) {},//Listen for events where the index value changes
getCurTab : function () {},//return x,y,z property
```

2.4.8 XML Syntax

Element

- Element corresponding widget.
- Element's attributes corresponding widget's attributes (0 value can be ignored).
- Text content is the widget instance's name.

```
<type a1="xx" a2="xx" a3="xx" a4="xx">name</type>
```

Nesting

The nesting structure of elements is consistent with the nesting structure of actual widgets.

```
<fatherType a1="xx" a2="xx" a3="xx" a4="xx">fatherName

    <childType a1="xx" a2="xx" a3="xx">childName1</childType>
    <childType a1="xx" a2="xx" a3="xx">childName2
        <childType a1="xx" a2="xx" a3="xx">childName3</childType>
        <childType a1="xx" a2="xx" a3="xx">childName4</childType>
    </childType>
</fatherType>
```

Specifications

Element	Attribu	Attribu	Attribu	Attribu	Attribu	Attribu	Attribu	Attribu	Attribu	Attribu	Attribu	Attribu	Attribu	Attribu	Attribu	Attribu	Attribu	Attributes17
win	x	y	w	h	hidden													
textbo:	x	y	w	h	text	font	fontSize	color	mode	inputat								
img	x	y	w	h	scale	scaleY	rotationAngle	blend Mod	opacity	file	folde	duration						
seekbar	x	y	w	h	folde	picture	orientation	centralX	centralY	startDegree	endDegree	reverse	blend Mod	opacity				
tabview	x	y	w	h	transition													
tab	x	y	w	h	idx	idy												
curtainview	x	y	w	h	transition													
curtain	x	y	w	h	scope	orientation	transition											
icon	x	y	w	h	font	picture	highlightPicture	fontColo	fontSize	text	textX	textY	pictureX	pictureY	mode	blend Mod	opacity	
script	file																	
switch	x	y	w	h	picture	highlightPicture	click Picture	click HighlightPicture	pictureX	pictureY	blend Mod	opacity	mode	duration				
page	x	y	w	h														
screen	w	h																
grid	x	y	rowNber	colNumber	rowGap	colGap												
gallery	x	y	w	h	folde	main	centerBg	centerPercent	sideS	sidePosPcent	blend Mod	opacity						
animateTransform	type	from	to	dur	repeatCour													
motorizedcurtain	x	y	w	h	switc	switc Close	switc Pause											
keyboard	ime																	

Attribute	Description	Values
x	Relative left coordinate	number
y	Relative top coordinate	number
w	Width	number
h	Height	number
hidden	Hidden	hidden
text	Text string	string
font	Font file	file path
fontSize	Font size	number
color	RGB hex color	#RRGGBB
mode(textbox)	Text effect	truncate, verticalscroll, horizontalscroll, transition
mode(icon)	Highlight effect on press	normal, fade, scale, fadeScale, array
mode(switch)	Highlight effect on press	array
inputable	Soft keyboard	boolean
scaleX	Horizontal scaling ratio	number
scaleY	Vertical scaling ratio	number
rotationAngle	Rotation angle	number
blendMode	Image blend mode	imgBypassMode, imgFilterBlack, imgSrcOverMode, imgCoverMode
opacity	Opacity from 0 to 255	number
file	File path	string
folder	Folder path	string
duration	Animation duration (milliseconds)	number
picture	Image file path	string
orientation(seekbar)	Orientation	vertical, V, horizontal, H, arc
orientation(curtain)	Direction	middle, up, down, left, right
centralX	Arc center x-coordinate	number
centralY	Arc center y-coordinate	number
startDegree	Arc starting angle	number
endDegree	Arc ending angle	number
transition	Transformation effect	normal, fade, scale, fadeScale
idx	Horizontal index	number
idy	Vertical index	number
scope	Range (from 0 to 1)	number
highlightPicture	Highlight image file path	string
fontColor	RGB hex color	#RRGGBB
textX	Relative x-coordinate of text	number
textY	Relative y-coordinate of text	number
pictureX	Relative x-coordinate of image	number
pictureY	Relative y-coordinate of image	number
rowNumber	Number of rows	number
colNumber	Number of columns	number
rowGap	Row spacing	number
colGap	Column spacing	number
mainBg	Main background image file path	string

continues on next page

Table 29 – continued from previous page

Attribute	Description	Values
centerBg	Center background image file path	string
centerPercent	Center area percentage	number
sideScale	Default scaling ratio for side images	number
sidePosPercent	Side image position percentage	number
type(animateTrans	Animation type	rotate
from	Starting value of animation	number
to(animateTransfo	End value of animation	number
dur	Animation duration	number
repeatCount	Number of animation repetitions	number
switchOpen	Motorized curtain open button name	string
switchClose	Motorized curtain close button name	string
pauseOpen	Motorized curtain pause button name	string
ime	Input method	null, pinyin
type(onClick)	Behavior type triggered by click event	jump, control
to(onClick)	Action target	light, multiLevel
id1	Main parameter	number
id2	Secondary parameter	number

Example

Win

```
<win
    x="0"
    y="0"
    w="480"
    h="480">cool_win
</win>
```

Img

```
<img
    x="80"
    y="70"
    w="303"
    h="239"
    opacity="255"
    file="app/box/resource/new_folder/aa2.bin"
    blendMode="imgFilterBlack"
    rotationAngle="0"
```

(continues on next page)

(continued from previous page)

```
    scaleX="1"
    scaleY="1">image3
</img>
```

2.4.9 Middleware

RVD exports the SaaA package. The firmware needs to parse and play it.

Package

Resource	XML	JavaScript
Pictures and font files, etc.	Describes the initial nested tree structure and specific parameters of the widget	Customized behaviors, such as triggering behaviors of widget gesture events, peripheral operations, printing logs, etc.

- Packages are in the `root/app` folder of File system image, and a launcher in firmware will iterate through these packages and set a start button on the screen for each package. Click the button to start the corresponding package.

Launcher

- The implementation of the launcher is in this file `realgui\SaaA\frontend_launcher.c`.
- It uses a grid widget to layout the apps' button. Then it iterates the `app` folder, to find all XML files, which represent apps.
- The launcher gets the title and icon of the APP, and use a button widget to display them. The click event of the registration button is to start the app.

XML

- The xml file in the APP package describes the initial nested tree structure and specific parameters of the widget.
- Using `realgui\3rd\ezXML` to convert xml to C language data format. Please refer to ezXML SourceForge for details.
- The implementation of the xml parser is in this file `realgui\DOM\XML_DOM.c`. You can read the syntax description on the `XML syntax` page.
- According to the syntax protocol, this function `foreach_create` uses a recursive strategy to traverse each tag of xml and map the tag to the widget, configure the tag's attributes to the widget.
- After the xml traversal is completed, a C-APP has actually been created in the firmware, which is no different from the result of directly using the C-APP api.
- Then the JavaScript file mentioned in xml will be executed.

JavaScript

- JavaScript describes Customized behaviors, such as triggering behaviors of widget gesture events, peripheral operations, printing logs, etc.
- Based on JerryScript engine on `realgui\3rd\js` for common syntax. Please refer to JerryScript for details.
- The implementation of the JavaScript parser is files starting with js in this folder `realgui\SaaA`. You can read the syntax description on the `JavaScript syntax` page.
- `DECLARE_HANDLER` is used to define a function as a C language implementation of a JavaScript function.
- `REGISTER_METHOD` and `REGISTER_METHOD_NAME` are used to add a function to a javascript object, so you can call it in script.
- In a javascript file, there are some variable definitions, function definitions, and function calls. When the app starts, as mentioned above, the JavaScript file will be executed at the end of the XML parsing, and the function calls in it will be executed, mainly some initialization behaviors and the registration of event listeners.
- The callback functions of those events will not be executed until the event occurs.

Example

Progressbar API

```
//Read and write the progress value of a progressbar tag called 'tag name'
progressbar.getElementById('tag name')
var progress = progressbar.progress(0.7)
```

Define A Progressbar Object

In fact, this object is added to the global object. Using property of the global object does not require explicitly calling the global object.

```
jerry_value_t progress = jerry_create_object();
js_set_property(global_obj, "progressbar", progress);
```

Add 2 Functions to The Progressbar Object

```
REGISTER_METHOD(progress, progress);
REGISTER_METHOD(progress, getElementById);
```

Define 2 Functions

- The `progress` is used to write and read the progressbar's progress.
- Input formal parameters are in the array `args`. The first in it is the progress number. If this parameter exists, which means that the progress needs to be set. Using `jerry_get_number_value()` to convert javascript parameter to c language variable.
- The return value is the progress you want to get, using `jerry_create_number` to convert c language variable to javascript variable. By the way, the form of these javascript variables in C language is an index of an unsigned integer.

```
DECLARE_HANDLER(progress)
{
    gui_obj_t *obj = NULL;
    jerry_get_object_native_pointer(this_value, (void *)&obj, NULL);
    if (args_cnt >= 1 && jerry_value_is_number(args[0]))
    {
        gui_progressbar_set_percentage((void *)obj, jerry_get_number_value(args[0]));
    }
    float per = gui_progressbar_get_percentage((void *)obj);
    return jerry_create_number(per);
}
```

- The `getElementById` is used to get the tag handle, refer to getElementById on MDN for more usage.
- Input formal parameter is the tag's specified name. Using `js_value_to_string` to convert JS form name to C form char array, and get the pointer handle, and assign value to tag. It is a little different from standard function definitions, which return the new instantiate tag.

```
DECLARE_HANDLER(getElementById)
{
    if (args_cnt != 1 || !jerry_value_is_string(args[0]))
    {
        return jerry_create_undefined();
    }
    jerry_value_t global_obj = jerry_get_global_object();
    jerry_value_t app_property = js_get_property(global_obj, "app");
    gui_app_t *app = NULL;
    jerry_get_object_native_pointer(app_property, (void *)&app, NULL);
    gui_obj_t *widget = NULL;
    char *a = js_value_to_string(args[0]);
    gui_obj_tree_get_widget_by_name(&app->screen, a, &widget);
    gui_free(a);
    jerry_set_object_native_pointer(this_value, widget, NULL);
    jerry_release_value(global_obj);
    jerry_release_value(app_property);
    return jerry_create_undefined();
}
```

Light Control

This page shows how the UI switch corresponds to the peripheral switch.

```
//IO P1_1 is set to low level
var P1_1 = 9
var LED1 = new Gpio(P1_1, 'out');
LED1.writeSync(0)
```

Light Switch Data

Data	Value type	Brief
gpio	number	index of light
direction	out / in	direction of signal
write value	number	0 for turning off / 1 for turning on

- Refer to onoff npm package usage for more information.

GPIO Light Switch

- Get gpio index, direction, and write value.
- Use gpio driver `drv_pin_mode()` & `drv_pin_write()` to operate it.

MATTER Light Switch

- Get gpio index, and write value.
- Transform data to matter protocol.
- Use `matter_send_msg_to_app()` to operate lights.

MESH Light Switch

- Get gpio index, and write value.
- Transform data to mesh protocol.
- Use `matter_send_msg_to_app()` to operate lights.

The following code example is the `writeSync`'s control light implementation for RTL87X2G. First get gpio value and direction value, then use specify driver API to operate light.

```
#ifdef RTL87x2G
#define ENABLE_MATTER_SWITCH
#define ENABLE_MESH_SWITCH
#define ENABLE_GPIO_SWITCH
#endif

#if defined ENABLE_GPIO_SWITCH
#include "rtl_gpio.h"
#include "rtl_rcc.h"
#include "drv_gpio.h"
#include "drv_i2c.h"
#include "drv_touch.h"
#include "drv_lcd.h"
#include "touch_gt911.h"
#include "string.h"
#include "trace.h"
#include "utils.h"
#endif
```

(continues on next page)

(continued from previous page)

```
#if defined ENABLE_MESH_SWITCH
#include "app_msg.h"
T_IO_MSG led_msg = {.type = IO_MSG_TYPE_LED_ON};
T_IO_MSG led_off_msg = {.type = IO_MSG_TYPE_LED_OFF};
#endif

#if defined ENABLE_MATTER_SWITCH
#endif

DECLARE_HANDLER(writeSync)
{
    gui_log("enter writeSync:%d\n", args[0]);
    if (args_cnt >= 1 && jerry_value_is_number(args[0]))
    {
        int write_value = jerry_get_number_value(args[0]);
        int gpio = -1;
        jerry_value_t v1;
        jerry_value_t v2;
        v1 = js_get_property(this_value, "gpio");
        v2 = js_get_property(this_value, "direction");
        gpio = jerry_get_number_value(v1);
        jerry_release_value(v1);
        char *direction = js_value_to_string(v2);
        jerry_release_value(v2);
        int mode = 0;

        if (gpio >= 0)
        {
            gui_log("gpio%d, %d, %d", gpio, mode, write_value);

            /**
             * GPIO
             */
            #ifdef ENABLE_GPIO_SWITCH
            if (!strcmp(direction, "out"))
            {
                mode = PIN_MODE_OUTPUT;
            }
            else if (!strcmp(direction, "in"))
            {
                mode = PIN_MODE_INPUT;
            }
            drv_pin_mode(gpio, mode);
            drv_pin_write(gpio, write_value);
            #endif

            /**
             * MESH
             */
            #ifdef ENABLE_MESH_SWITCH
            extern bool app_send_msg_to_apptask(T_IO_MSG *p_msg);
            if(write_value == 0){
                led_msg.u.param = 0x64+gpio;
                app_send_msg_to_apptask(&led_msg);}
            else
            {
                led_off_msg.u.param = 0x64+gpio;
```

(continues on next page)

(continued from previous page)

```
                app_send_msg_to_apptask(&led_off_msg);
            }
            #endif

            /**
             * MATTER
             */
            #ifdef ENABLE_MATTER_SWITCH
            if (gpio >= 0)
            {
                extern bool matter_send_msg_to_app(uint16_t sub_type, uint32_t param);
                uint32_t param = gpio << 8 | write_value;
                if (gpio != 49052)
                {
                    //single
                    matter_send_msg_to_app(0, param);
                }
                else
                {
                    //group
                    matter_send_msg_to_app(1, param);
                }
            #endif
        }

        gui_free(direction);
    }
    return jerry_create_undefined();
}
```

CHAPTER

THREE

WIDGETS

Table 1: Abbreviation

Words	Definition
acc	Accelerate
addr	Address
att	Attribute
ax	Absolute coordinates on the x-axis
blit	Bit-Block Image Transfer
buff	Buffer
cb	Callback
cbsize	Cubesize
ctor	Constructor
cur_idx	Current index in x direction
cur_idy	Current index in y direction
cx	Center coordinates on the x-axis
dc	Display Canvas
dur	Duration
dx	The difference along the x-axis for touchpad
fd	File Descriptor
fg	Foreground
fs	File System
hw	Hardware
id	Index
img	Image
info	Information
init	Initialize
mem	Memory
mq	Message queue
nz	Normal vector in Z direction of plane
obj	Object
off	Offset
pic	Picture
pos	Position
prev	Previous
rst	Result
src	Source
sx	Scale in x direction
tmp	Temporary
tx	Translation in x direction

3.1 Obj

The Object implements the basic properties of widgets on a screen. The screen widget is the root node of a widget tree. The screen coordinate system is set as follows. The origin of the polar coordinates is the negative direction of the Y axis, and the positive direction of the polar coordinates is clockwise:

3.1.1 Usage

Table 2: Gui_Obj Table

Description	API
Get the root object	*gui_obj_get_root()*
Create object	*gui_obj_create()*
Add event	gui_obj_add_event_cb
Set event	gui_obj_enable_event
Free the widget tree recursively from the root to the leaves	gui_obj_tree_free
Print the widget tree recursively from the root to the leaves	gui_obj_tree_print
Get the count of one type of widget in the tree	gui_obj_tree_count_by_type
Hide/Show widget	gui_obj_tree_show
Enable object show or not	*gui_obj_show()*
Get the root object of tree	gui_obj_tree_get_root
Get the child object of tree	gui_obj_get_child_handle
Judge the object if in range of the rect	*gui_obj_in_rect()*
Skip all actions of the parent object(left/right/down/up slide hold actions)	• gui_obj_skip_all_parent_left_hold • gui_obj_skip_all_parent_right_hold • gui_obj_skip_all_parent_down_hold • gui_obj_skip_all_parent_up_hold
Skip all actions of the child object(left/right/down/up slide hold actions)	• gui_obj_skip_all_child_left_hold • gui_obj_skip_all_child_right_hold • gui_obj_skip_all_child_down_hold • gui_obj_skip_all_child_up_hold
Skip actions of the other object(left/right/down/up slide hold actions)	• gui_obj_skip_other_left_hold • gui_obj_skip_other_right_hold • gui_obj_skip_other_down_hold • gui_obj_skip_other_up_hold
Get area of the object	*gui_obj_get_area()*
Point-in-Rectangle Range Check	*gui_obj_point_in_obj_rect()*
CRC check	*gui_obj_checksum()*
Get widget in tree by name	gui_obj_tree_get_widget_by_name
Get widget in tree by type	gui_obj_tree_get_widget_by_type
Update animate	animate_frame_update
Set animate	gui_obj_set_animate
Print the tree in a breadth-first search manner	gui_obj_tree_print_bfs

3.1.2 API

Functions

gui_obj_t *__gui_obj_get_root__(void)

Get the root GUI object.

This function returns a pointer to the root GUI object in the widget tree.

Returns
A pointer to the root GUI object.

gui_obj_t *__gui_obj_get_fake_root__(void)

Get the fake_root GUI object, which would not be drawn.

This function returns a pointer to the fake_root GUI object in the widget tree.

Returns
A pointer to the fake_root GUI object.

gui_obj_t *__gui_obj_create__(void *parent, const char *name, int16_t x, int16_t y, int16_t w, int16_t h)

creat an obj widget.

Parameters

- **parent** – the father widget it nested in.
- **filename** – the obj widget name.
- **x** – the X-axis coordinate of the widget.
- **y** – the Y-axis coordinate of the widget.
- **w** – the width of the widget.
- **h** – the hight of the widget.

Returns
gui_obj_t*.

void __gui_obj_show__(void *obj, bool enable)

set object show or not.

Parameters

- **obj** – the root of the widget tree.
- **enable** – true for show, false for hide.
  - **Example usage**

```
static void app_main_task(gui_app_t *app)
{
    gui_img_t *hour;
    gui_obj_show(hour,false);
    gui_obj_show(hour,true);

}
```

bool __gui_obj_out_screen__(gui_obj_t *obj)

judge the obj if out of screen.

void **gui_obj_get_clip_rect**(gui_obj_t *obj, gui_rect_t *rect)

Calculate the clipping rectangle of a GUI object relative to its top-level ancestor.

Parameters

- **obj** – The GUI object for which the clipping rectangle is calculated.
- **rect** – The output rectangle that will contain the calculated clipping area.

bool **gui_obj_in_rect**(gui_obj_t *obj, int16_t x, int16_t y, int16_t w, int16_t h)

judge the obj if in range of this_widget rect.

Parameters

- **obj** – pointer to the GUI object.
- **x** – the X-axis coordinate of the widget.
- **y** – the Y-axis coordinate of the widget.
- **w** – the width of the widget.
- **h** – the hight of the widget.

Returns
true.

Returns
false.

void **gui_obj_enable_this_parent_short**(gui_obj_t *obj)

enable all short click actions from parent object to the root object.

enable all long press actions from parent object to the root object.

Parameters
obj – the root of the widget tree.

void **gui_obj_get_area**(gui_obj_t *obj, int16_t *x, int16_t *y, int16_t *w, int16_t *h)

get the area of this_widget obj.

Parameters

- **obj** – pointer to the GUI object.
- **x** – the X-axis coordinate of the widget.
- **y** – the Y-axis coordinate of the widget.
- **w** – the width of the widget.
- **h** – the hight of the widget.

bool **gui_obj_point_in_obj_rect**(gui_obj_t *obj, int16_t x, int16_t y)

judge the point if in range of this_widget obj rect.

Parameters

- **obj** – widget object pointer.
- **x** – the X-axis coordinate.
- **y** – the Y-axis coordinate.

Returns
true.

Returns
false.

bool **gui_obj_point_in_obj_circle**(gui_obj_t *obj, int16_t x, int16_t y)

judge the point if in range of this_widget obj circle.

Parameters

- **obj** – widget object pointer.
- **x** – the X-axis coordinate.
- **y** – the Y-axis coordinate.

Returns
true.

Returns
false.

uint8_t **gui_obj_checksum**(uint8_t seed, uint8_t *data, uint8_t len)

do crc check.

Parameters

- **seed** – the initial value to start the checksum calculation.
- **data** – pointer to the array of bytes for which the checksum is to be calculated.
- **len** – the number of bytes in the array.

Returns
uint8_t.

gui_obj_t ***gui_get_root**(gui_obj_t *object)

print name by bfs order.

Parameters
object – widget pointer.

Returns
gui_obj_t * root.

void **gui_obj_absolute_xy**(gui_obj_t *obj, int *absolute_x, int *absolute_y)

calculate the absolute coordinates of a GUI object.

This function calculates the absolute (global) X and Y coordinates of a given GUI object based on its local position within the parent hierarchy.

Note: This function assumes that `obj` is a valid pointer and that `absolute_x` and `absolute_y` are valid pointers to integers.

Parameters

- **obj** – pointer to the GUI object for which to calculate absolute coordinates.
- **absolute_x** – pointer to an integer where the absolute X coordinate will be stored.
- **absolute_y** – pointer to an integer where the absolute Y coordinate will be stored.

void **gui_obj_hidden**(gui_obj_t *obj, bool hidden)

set the visibility of a GUI object.

This function sets the visibility of a given GUI object by adjusting its hidden state.

Parameters

- **obj** – pointer to the GUI object that will be updated.
- **hidden** – boolean flag indicating whether the object should be hidden (true) or shown (false).

const char ***gui_widget_name**(gui_obj_t *widget, const char *name)

set or retrieve the name of a GUI widget.

This function sets the name of a given GUI widget if the provided name is valid. It returns the current name of the widget.

Parameters

- **widget** – pointer to the GUI widget whose name will be set or retrieved.
- **name** – pointer to a string containing the new name for the widget. If the name is valid, it will be set as the widget's name.

Returns

the current name of the widget.

void **gui_update_speed**(int *speed, int speed_recode[])

update touch pad speed vertical.

This function updates the current speed and records the speed change history.

Parameters

- **speed** – pointer to the current speed, which will be updated by the function.
- **speed_recode** – array to record speed changes, which will be updated by the function.

void **gui_inertial**(int *speed, int end_speed, int *offset)

inertial calculation.

This function performs inertial calculations based on the current speed, end speed, and offset.

Parameters

- **speed** – pointer to the current speed, which will be updated by the function.
- **end_speed** – target end speed.
- **offset** – pointer to the offset, which will be updated by the function.

uint32_t **gui_get_obj_count**(void)

get widget count.

void **gui_set_location**(gui_obj_t *obj, uint16_t x, uint16_t y)

Set the location of a GUI object.

This function sets the X and Y coordinates of the specified GUI object.

Parameters

- **obj** – Pointer to the GUI object to set location for.
- **x** – The X coordinate to set.
- **y** – The Y coordinate to set.

void **gui_dom_create_tree_nest**(const char *xml, gui_obj_t *parent_widget)

API to create a widget tree structure from an XML file and associate it with a parent widget.

Parameters

- **xml** – The path to the XML file to be parsed.
- **parent_widget** – The parent widget to which the tree structure is to be associated.

char ***gui_dom_get_preview_image_file**(const char *xml)

Extracts the preview image file path from an XML file.

This function parses the given XML file and attempts to find the preview image file path by looking for specific tags within the XML.

Parameters

xml_file – The path to the XML file to be parsed.

Returns

A string containing the path to the preview image file. If the XML file cannot be loaded or the preview image file path cannot be found, returns NULL.

void **gui_update_speed_by_displacement**(int *speed, int speed_recode[], int displacement)

Update the speed based on displacement.

This function updates the speed value based on the given displacement. It also uses a speed record array to achieve this.

Parameters

- **speed** – Pointer to the speed variable to update.
- **speed_recode** – Array holding the speed records.
- **displacement** – The displacement value to consider for speed update.

void **gui_obj_move**(gui_obj_t *obj, int x, int y)

Move a widget object to specified coordinates.

This function moves the specified widget object to a new (x, y) coordinate position.

Parameters

- **obj** – Pointer to the widget object to be moved.
- **x** – The new x-coordinate for the widget object.
- **y** – The new y-coordinate for the widget object.

void **gui_obj_create_timer**(gui_obj_t *obj, uint32_t interval, bool reload, void (*callback)(void*))

Set a timer for a GUI object.

This function sets a timer for the specified GUI object with a given interval. The timer can be configured to reload automatically or run only once. When the timer expires, the provided callback function is called.

Parameters

- **obj** – Pointer to the GUI object to set the timer for.
- **interval** – The interval in milliseconds for the timer.
- **reload** – Boolean flag indicating whether the timer should reload automatically (true) or run only once (false).
- **callback** – Pointer to the callback function to be called when the timer expires.

void **gui_obj_delete_timer**(gui_obj_t *obj)

void **gui_obj_start_timer**(gui_obj_t *obj)

void **gui_obj_stop_timer**(gui_obj_t *obj)

3.2 Img

The image widget is the basic widget used to display images. Image widgets support moving, zooming, rotating, etc.

3.2.1 Usage

Create Widget

It is possible to use *gui_img_create_from_mem()* to create an image widget from memory, or use *gui_img_create_from_fs()* to create an image widget from a file. Alternatively, *gui_img_create_from_ftl()* can be used to create an image widget from ftl. If the width or height of the image widget is set to *0*, the widget's size will be set according to the size of the image source automatically.

Update Location

If it is necessary to update the location of an image widget, use *gui_img_set_location()* to relocate.

Set Attribute

It is possible to use *gui_img_set_attribute()* to set the attribute of an image widget, replace it with a new image, and set a new coordinate.

Get Height/Width

If you want to get the height/width of image widget, do so with *gui_img_get_height()* or *gui_img_get_width()*.

Refresh

Refresh the image size using *gui_img_refresh_size()*.

Blend Mode

Set the image's blend mode using *gui_img_set_mode()*.

Translation

Use *gui_img_translate()* to move the image widget. It can move an image widget to a new coordinate without changing the original coordinate in the widget's attribute.

Rotation

Rotate the image widget around the center of the circle with this API *gui_img_rotation()*.

Zoom

You can adjust the size of the image widget to fit your requirements by this API *gui_img_scale()*.

Opacity

The opacity value of the image is adjustable, and it can be set using *gui_img_set_opacity()*.

Animation

The *gui_img_set_animate()* can be used to set the animation effects for the image widget.

Quality

The image's quality can be set using *gui_img_set_quality()*.

Screenshot

The *gui_img_tree_convert_to_img()* can be used to save a fullscreen screenshot. The saved image will be in RGB format.

3.2.2 Example

```
#include "root_image_hongkong/ui_resource.h"
#include "gui_img.h"
#include "gui_text.h"
#include "draw_font.h"

char *tbl_text = "gui_img_create_from_mem";

void page_tbl(void *parent)
{
    static char array1[50];
    static char array2[50];

    gui_set_font_mem_resourse(24, TEST_FONT24_DOT_BIN, TEST_FONT24_TABLE_BIN);

    gui_img_t *img_test = gui_img_create_from_mem(parent, "test", SET_ON_BIN, 0, 0, 0,
↪ 0);
```

(continues on next page)

(continued from previous page)

```
    gui_text_t *text1 = gui_text_create(parent, "text1", 10, 100, 300, 30);
    gui_text_set(text1, tb1_text, GUI_FONT_SRC_BMP, 0xffffffff, strlen(tb1_text), 24);
    gui_text_mode_set(text1, LEFT);

    gui_text_t *text2 = gui_text_create(parent, "text2", 10, 130, 330, 30);
    gui_text_set(text2, tb1_text, GUI_FONT_SRC_BMP, 0xffffffff, strlen(tb1_text), 24);
    gui_text_mode_set(text2, LEFT);
    sprintf(array1, "gui_img_get_height %d", gui_img_get_height(img_test));
    text2->utf_8 = array1;
    text2->len = strlen(array1);

    gui_text_t *text3 = gui_text_create(parent, "text3", 10, 160, 330, 30);
    gui_text_set(text3, tb1_text, GUI_FONT_SRC_BMP, 0xffffffff, strlen(tb1_text), 24);
    gui_text_mode_set(text3, LEFT);
    sprintf(array2, "gui_img_get_width %d", gui_img_get_width(img_test));
    text3->utf_8 = array2;
    text3->len = strlen(array2);
}

void page_tb2(void *parent)
{
    gui_set_font_mem_resourse(24, TEST_FONT24_DOT_BIN, TEST_FONT24_TABLE_BIN);

    gui_img_t *img_test = gui_img_create_from_mem(parent, "test", SET_ON_BIN, 0, 0, 0,
↪ 0);
    gui_img_set_location(img_test, 50, 50);

    gui_text_t *text2 = gui_text_create(parent, "text2", 10, 100, 330, 24);
    gui_text_set(text2, "gui_img_set_location", GUI_FONT_SRC_BMP, 0xffffffff, 20, 24);
    gui_text_mode_set(text2, LEFT);
}

void page_tb3(void *parent)
{
    gui_img_t *img_test = gui_img_create_from_mem(parent, "test", SET_ON_BIN, 0, 0, 0,
↪ 0);
    gui_img_set_attribute(img_test, "test", SET_OFF_BIN, 20, 20);

    gui_text_t *text3 = gui_text_create(parent, "text3", 10, 100, 330, 24);
    gui_text_set(text3, "gui_img_set_attribute", GUI_FONT_SRC_BMP, 0xffffffff, 21,␣
↪24);
    gui_text_mode_set(text3, LEFT);

}

void page_tb4(void *parent)
{
    gui_set_font_mem_resourse(24, TEST_FONT24_DOT_BIN, TEST_FONT24_TABLE_BIN);

    gui_img_t *img_test = gui_img_create_from_mem(parent, "test", SET_ON_BIN, 0, 0, 0,
↪ 0);
    gui_img_scale(img_test, 0.5, 0.5);

    gui_text_t *text4 = gui_text_create(parent, "text4", 10, 100, 330, 24);
    gui_text_set(text4, "gui_img_scale", GUI_FONT_SRC_BMP, 0xffffffff, 13, 24);
    gui_text_mode_set(text4, LEFT);
}
```

(continues on next page)

(continued from previous page)

```
void page_tb5(void *parent)
{
    gui_set_font_mem_resourse(24, TEST_FONT24_DOT_BIN, TEST_FONT24_TABLE_BIN);

    gui_img_t *img_test = gui_img_create_from_mem(parent, "test", SET_ON_BIN, 0, 0, 0,
↪ 0);
    gui_img_translate(img_test, 100, 100);

    gui_text_t *text5 = gui_text_create(parent, "text5", 10, 100, 330, 24);
    gui_text_set(text5, "gui_img_translate", GUI_FONT_SRC_BMP, 0xffffffff, 17, 24);
    gui_text_mode_set(text5, LEFT);
}

void page_tb6(void *parent)
{
    gui_set_font_mem_resourse(24, TEST_FONT24_DOT_BIN, TEST_FONT24_TABLE_BIN);

    gui_img_t *img_test = gui_img_create_from_mem(parent, "test", SET_ON_BIN, 0, 0, 0,
↪ 0);
    gui_img_rotation(img_test, 10, 0, 0);

    gui_text_t *text6 = gui_text_create(parent, "text6", 10, 100, 330, 24);
    gui_text_set(text6, "gui_img_rotation", GUI_FONT_SRC_BMP, 0xffffffff, 16, 24);
    gui_text_mode_set(text6, LEFT);
}
```

3.2.3 API

Functions

uint16_t **gui_img_get_width**(*gui_img_t* *_this)

load the image to read it's width.

Parameters
_this – the image widget pointer.

Returns
uint16_t image's width.

uint16_t **gui_img_get_height**(*gui_img_t* *_this)

load the image to read it's hight.

Parameters
_this – the image widget pointer.

Returns
uint16_t image's height.

void **gui_img_refresh_size**(*gui_img_t* *_this)

refresh the image size from src.

Parameters
_this – the image widget pointer.

void **gui_img_set_location**(*gui_img_t* *_this, uint16_t x, uint16_t y)

set the image's location.

Parameters

- **_this** – the image widget pointer.
- **x** – the x coordinate.
- **y** – the y coordinate.

void **gui_img_set_mode**(*gui_img_t* *_this, BLEND_MODE_TYPE mode)

set the image's blend mode.

Parameters

- **_this** – the image widget pointer.
- **mode** – the enumeration value of the mode is BLEND_MODE_TYPE.

void **gui_img_set_attribute**(*gui_img_t* *_this, const char *name, void *addr, int16_t x, int16_t y)

set x,y and file path.

Parameters

- **_this** – image widget.
- **name** – change widget name.
- **addr** – change picture address.
- **x** – X-axis coordinate.
- **y** – Y-axis coordinate.

void **gui_img_rotation**(*gui_img_t* *_this, float degrees, float c_x, float c_y)

rotate the image around the center of the circle.

Parameters

- **_this** – the image widget pointer.
- **degrees** – clockwise rotation absolute angle.
- **c_x** – the X-axis coordinates of the center of the circle.
- **c_y** – the Y-axis coordinates of the center of the circle.

void **gui_img_scale**(*gui_img_t* *_this, float scale_x, float scale_y)

change the size of the image, take (0, 0) as the zoom center.

Parameters

- **_this** – the image widget pointer.
- **scale_x** – scale in the x direction.
- **scale_y** – scale in the y direction.

void **gui_img_translate**(*gui_img_t* *_this, float t_x, float t_y)

move image.

Parameters

- **_this** – the image widget pointer.
- **t_x** – new X-axis coordinate.

- **t_y** – new Y-axis coordinate.

void **gui_img_skew_x**(*gui_img_t* *_this, float degrees)

skew image on X-axis.

Parameters

- **_this** – the image widget pointer.
- **degrees** – skew angle.

void **gui_img_skew_y**(*gui_img_t* *_this, float degrees)

skew image on Y-axis.

Parameters

- **_this** – the image widget pointer.
- **degrees** – skew angle.

void **gui_img_set_opacity**(*gui_img_t* *_this, unsigned char opacity_value)

add opacity value to the image.

Parameters

- **_this** – the image widget pointer.
- **opacity_value** – The opacity value ranges from 0 to 255, default 255.

gui_img_t ***gui_img_create_from_mem**(void *parent, const char *name, void *addr, int16_t x, int16_t y, int16_t w, int16_t h)

creat an image widget from memory address.

Note: creat an image widget and set attribute.

Parameters

- **parent** – the father widget it nested in.
- **name** – widget name.
- **addr** – bin file address.
- **x** – the X-axis coordinate of the widget.
- **y** – the Y-axis coordinate of the widget.
- **w** – the width of the widget.
- **h** – the hight of the widget.

Returns

return the widget object pointer.

gui_img_t ***gui_img_create_from_ftl**(void *parent, const char *name, void *ftl, int16_t x, int16_t y, int16_t w, int16_t h)

creat an image widget from memory address.

Note: creat an image widget and set attribute.

Parameters

- **parent** – the father widget it nested in.
- **name** – widget name.
- **ftl** – not xip address, use ftl address.
- **x** – the X-axis coordinate of the widget.
- **y** – the Y-axis coordinate of the widget.
- **w** – the width of the widget.
- **h** – the hight of the widget.

Returns
return the widget object pointer.

gui_img_t ***gui_img_create_from_fs**(void *parent, const char *name, void *file, int16_t x, int16_t y, int16_t w, int16_t h)

creat an image widget from filesystem.

Parameters

- **parent** – the father widget it nested in.
- **name** – image widget name.
- **file** – image file path.
- **x** – the X-axis coordinate of the widget.
- **y** – the Y-axis coordinate of the widget.
- **w** – the width of the widget.
- **h** – the hight of the widget.

Returns
gui_img_t*.

void **gui_img_set_animate**(*gui_img_t* *_this, uint32_t dur, int repeat_count, void *callback, void *p)

set animate.

Parameters

- **_this** – pointer.
- **dur** – animation time cost in ms.
- **repeat_count** – rounds to repeat.
- **callback** – every frame callback.
- **p** – callback's parameter.

void **gui_img_set_quality**(*gui_img_t* *_this, bool high_quality)

set the image's quality.

Parameters

- **_this** – the image widget pointer.
- **high_quality** – image drawn in high quality or not.

void **gui_img_tree_convert_to_img**(gui_obj_t *obj, gui_matrix_t *matrix, uint8_t *shot_buf)

convert a tree to a image data.

Parameters

- **obj** – tree root.
- **matrix** – null if no need to transform.

float **gui_img_get_transform_scale_x**(*gui_img_t* *img)

get the transform scale in the X direction for a GUI image.

Parameters
img – pointer to the GUI image object.

Returns
the scale in the X direction.

float **gui_img_get_transform_scale_y**(*gui_img_t* *img)

get the transform scale in the Y direction for a GUI image.

Parameters
img – pointer to the GUI image object.

Returns
the scale in the Y direction.

float **gui_img_get_transform_degrees**(*gui_img_t* *img)

get the rotation angle in degrees for a GUI image.

Parameters
img – pointer to the GUI image object.

Returns
the rotation angle in degrees.

float **gui_img_get_transform_c_x**(*gui_img_t* *img)

get the center X coordinate for rotate of a GUI image.

Parameters
img – pointer to the GUI image object.

Returns
the center X coordinate for transformations.

float **gui_img_get_transform_c_y**(*gui_img_t* *img)

get the center Y coordinate for rotate of a GUI image.

Parameters
img – pointer to the GUI image object.

Returns
the center Y coordinate for transformations.

float **gui_img_get_transform_t_x**(*gui_img_t* *img)

get the translation in the X direction for a GUI image.

Parameters
img – pointer to the GUI image object.

Returns
the translation in the X direction.

float **gui_img_get_transform_t_y**(*gui_img_t* *img)

get the translation in the Y direction for a GUI image.

Parameters
img – pointer to the GUI image object.

Returns
the translation in the Y direction.

void **gui_img_set_image_data**(*gui_img_t* *widget, const uint8_t *image_data_pointer)

Sets the image data for a specified image widget.

This function assigns the given image data to the specified image widget. The image data might correspond to various formats, and the format should be compatible with the handling of *gui_img_t*.

Parameters

- **widget** – The pointer to the image widget (*gui_img_t*) for which the image data is to be set.
- **image_data_pointer** – The pointer to the image data to be set to the widget. The data should persist as long as the widget needs it or until it is explicitly updated.

const uint8_t ***gui_img_get_image_data**(*gui_img_t* *widget)

Gets the image data from a specified image widget.

This function returns the current image data that is set in the specified image widget.

Parameters
widget – The pointer to the image widget (*gui_img_t*) from which the image data should be retrieved.

Returns
A pointer to the image data currently set in the widget. If no image data is set, the result may be `NULL`.

struct **gui_img_transform_t**

image widget structure

Public Members

float **degrees**

float gui_img_get_transform_degrees(gui_img_t *img);

float **c_x**

center of image x; float gui_img_get_transform_c_x(gui_img_t *img);

float **c_y**

center of image y; float gui_img_get_transform_c_y(gui_img_t *img);

float **scale_x**

float gui_img_get_transform_scale_x(gui_img_t *img);

float **scale_y**

float gui_img_get_transform_scale_y(gui_img_t *img);

float **t_x**

translate of screen x; float gui_img_get_transform_t_x(gui_img_t *img);

float **t_y**

translate of screen y; float gui_img_get_transform_t_y(gui_img_t *img);

float **t_x_old**

float **t_y_old**

struct **gui_img_t**

Public Members

gui_obj_t **base**

draw_img_t ***draw_img**

gui_img_transform_t ***transform**

void ***data**

void ***filename**

void ***ftl**

union gui_img_t

gui_animate_t ***animate**

uint32_t **opacity_value**

uint32_t **blend_mode**

uint32_t **src_mode**

uint32_t **high_quality**

uint32_t **press_flag**

press to change picture to the highlighted

uint32_t **release_flag**

uint32_t **need_clip**

uint8_t **checksum**

uint8_t **animate_array_length**

3.3 Text

The text widget is the basic widget used to display text, which can be used to output text in different fonts, different colors, and different sizes to the screen. In order to draw text, the font file can be a standard .ttf file or a customized .bin file.

3.3.1 Features

Text widgets can support the following features.

- UTF-8/UTF16/UTF-32 support
- Multi language support
- Text typesetting support
- Word wrap and texts scrolling
- Anti-aliasing
- Multi fonts support
- Multi font sizes support
- Thirty-two bit true color support
- Emoji support
- Custom animation effects support
- Standards TTF file support[①]
- Self-developed font files support

[①]: Only part of the chip support this feature.

3.3.2 Usage

Using functions to load font files and display text.

Initialize the Font File

In order to draw text, font files containing glyph information need to be loaded into the system.

The font file can be a standard .ttf file or a customized .bin file. The font file can be initialized ahead of time to avoid having to set the font type for each text widget.

- To initialize the new version customized bin font file, use `gui_font_mem_init(uint8_t *font_bin_addr)`.
- To initialize the standard TTF file to draw text, use `gui_font_stb_init(void *font_ttf_addr)`.

All customized bin font files are available from RTK technicians.

`FONT_BIN`, `FONT_TTF` are all addresses of the files stored in flash.

Create Text Widget

To create a text widget, you can use *`gui_text_create()`*, The coordinates on the screen and text box size have been identified after create. These attributes also can be modified whenever you want.

Note: The size of the text box should be larger than the string to be shown; out-of-range text will be hidden.

Set Text Attributes

Set Text

To add some texts or characters to a text widget and set text attributes with *`gui_text_set()`*.

Note: The text length must be the same as the set character length, and the font size of the text must be the same as the size of the loaded font file.

Font Type

The text widget support type setting. This function can be used to set the type. The type is a bin/ttf file address *`gui_text_type_set()`*.

Text Content

This interface can be used to set the content that needs to be displayed by the text widget *gui_text_content_set()*.

Text Encoding

The text widget supports input formats in UTF-8, UTF-16, and UTF-32 encodings simultaneously. Developers can use *gui_text_encoding_set()* to change the encoding format.

Convert to Img

By using this function *gui_text_convert_to_img()*, the text in the text widget will be converted into an image, stored in memory, and rendered using the image. It also supports image transformations such as scaling and rotation. This only applies to bitmap fonts.

Note: Because the content and font size information of the text widget is needed, it should be called after set text. If the content, font size, position, and other attributes of the text have been modified, you need to reuse this interface for conversion.

Text Input

Text widget supports the input setting. You can use this function to set input *gui_text_input_set()*.

Text Click

Text widget supports click. You can use this function to add the click event for text *gui_text_click()*.

Text Mode

Text widget supports seven typesetting modes. To set text typesetting mode, use: *gui_text_mode_set()*.

All type setting modes are as follows.

Table 3: Text Mode

Type	Line Type	X Direction	Y Direction	Widget
LEFT	Single-line	Left	Top	Text widget (Default)
CENTER	Single-line	Center	Top	Text widget
RIGHT	Single-line	Right	Top	Text widget
MULTI_LEFT	Multi-line	Left	Top	Text widget
MULTI_CENTER	Multi-line	Center	Top	Text widget
MULTI_RIGHT	Multi-line	Right	Top	Text widget
MID_LEFT	Multi-line	Left	Mid	Text widget
MID_CENTER	Multi-line	Center	Mid	Text widget
MID_RIGHT	Multi-line	Right	Mid	Text widget
SCROLL_X	Single-line	Right to Left	Top	Scroll text widget
SCROLL_Y	Multi-line	Left	Bottom to Top	Scroll text widget
SCROLL_Y_REVERSE	Multi-line	Right	Top to Bottom	Scroll text widget
VERTICAL_LEFT	Multi-line	Left	Top to Bottom	Text widget
VERTICAL_RIGHT	Multi-line	Right	Bottom to Top	Text widget

```
typedef enum
{
    /* TOP */
    LEFT                = 0x00,
    CENTER              = 0x01,
    RIGHT               = 0x02,
    MULTI_LEFT          = 0x03,
    MULTI_CENTER        = 0x04,
    MULTI_RIGHT         = 0x05,
    /* MID */
    MID_LEFT            = 0x10,
    MID_CENTER          = 0x11,
    MID_RIGHT           = 0x12,
    /* SCROLL */
    SCROLL_X            = 0x30,
    SCROLL_Y            = 0x31,
    SCROLL_Y_REVERSE    = 0x32,
    SCROLL_X_REVERSE    = 0x33,
    /* VERTICAL */
    VERTICAL_LEFT       = 0x40,
    VERTICAL_RIGHT      = 0x41,
} TEXT_MODE;
```

Text Move

It is possible to use this function *gui_text_move()* to move text to a specified location, but x and y cannot be larger than w and h of the text.

Set Animate

Using this function *gui_text_set_animate()* to set the animation and implement the animation effect in the corresponding callback function.

3.3.3 Example

Multiple Text Widget

```
#include "string.h"
#include "gui_obj.h"
#include "guidef.h"
#include "gui_text.h"
#include "draw_font.h"
#include "gui_app.h"
#include "rtk_gui_resource.h"

static char chinese[6] =
{
    0xE4, 0xB8, 0xAD,
    0xE6, 0x96, 0x87
};
static void app_launcher_ui_design(gui_app_t *app)
{
    gui_font_mem_init(HARMONYOS_SIZE24_BITS1_FONT_BIN);
    gui_font_mem_init(HARMONYOS_SIZE16_BITS4_FONT_BIN);
    gui_font_mem_init(HARMONYOS_SIZE32_BITS1_FONT_BIN);
    gui_font_mem_init(SIMKAI_SIZE24_BITS4_FONT_BIN);

    void *screen = &app->screen;

    gui_text_t *text1 = gui_text_create(screen,  "text1",  10, 10, 100, 50);
    gui_text_set(text1, chinese, GUI_FONT_SRC_BMP, APP_COLOR_WHITE, strlen(chinese),␣
↪24);
    gui_text_type_set(text1, HARMONYOS_SIZE24_BITS1_FONT_BIN, FONT_SRC_MEMADDR);
    gui_text_mode_set(text1, LEFT);

    gui_text_t *text2 = gui_text_create(screen,  "text2",  0, 50, 300, 50);
    gui_text_set(text2, "english", GUI_FONT_SRC_BMP, APP_COLOR_RED, 7, 16);
    gui_text_type_set(text2, HARMONYOS_SIZE16_BITS4_FONT_BIN, FONT_SRC_MEMADDR);
    gui_text_mode_set(text2, LEFT);

    char *string = "TEXT_WIDGET";
    gui_text_t *text3 = gui_text_create(screen,  "text3",  0, 90, 300, 50);
    gui_text_set(text3, string, GUI_FONT_SRC_BMP, APP_COLOR_BLUE, strlen(string), 32);
    gui_text_type_set(text3, HARMONYOS_SIZE32_BITS1_FONT_BIN, FONT_SRC_MEMADDR);
    gui_text_mode_set(text3, CENTER);

    gui_text_t *text4 = gui_text_create(screen,  "text4",  0, 150, 100, 200);
```

(continues on next page)

(continued from previous page)

```
    gui_text_set(text4, "ABCDEFGHIJKLMNOPQRSTUVWXYZ", GUI_FONT_SRC_BMP, gui_rgb(0,␣
↪0xff, 0xff), 24, 24);
    gui_text_type_set(text4, SIMKAI_SIZE24_BITS4_FONT_BIN, FONT_SRC_MEMADDR);
    gui_text_mode_set(text4, MULTI_CENTER);
}
```

Animate Text Widget

```
#include "root_image_hongkong/ui_resource.h"
#include "string.h"
#include "gui_obj.h"
#include "guidef.h"
#include "gui_text.h"
#include "draw_font.h"

void change_text_cb(gui_text_t *obj)
{
    if (obj->animate->current_frame > 0 && obj->animate->current_frame < 50)
    {
        gui_text_move(obj, 50, 150);
        gui_text_content_set(obj, "123456789", 9);
    }
    else if (obj->animate->current_frame > 50 && obj->animate->current_frame < 100)
    {
        gui_text_move(obj, 200, 150);
        gui_text_content_set(obj, "987654321", 9);

    }
    else
    {
        gui_text_move(obj, 125, 50);
        gui_text_content_set(obj, "abcdefghi", 9);
    }
}

void page_tb_activity(void *parent)
{
    gui_font_mem_init(SIMKAI_SIZE24_BITS4_FONT_BIN);

    gui_text_t *text = gui_text_create(parent,  "text",  0, 0, 100, 200);
    gui_text_set(text, "ABCDEFGHI", GUI_FONT_SRC_BMP, APP_COLOR_RED, 9, 24);
    gui_text_type_set(text, SIMKAI_SIZE24_BITS4_FONT_BIN, FONT_SRC_MEMADDR);
    gui_text_mode_set(text, MULTI_CENTER);
    gui_text_set_animate(text, 5000, 15, change_text_cb, text);
}
```

3.3.4 API

Enums

enum **TEXT_MODE**

Values:

enumerator **LEFT**

enumerator **CENTER**

enumerator **RIGHT**

enumerator **MULTI_LEFT**

enumerator **MULTI_CENTER**

enumerator **MULTI_RIGHT**

enumerator **MID_LEFT**

enumerator **MID_CENTER**

enumerator **MID_RIGHT**

enumerator **SCROLL_X**

enumerator **SCROLL_Y**

enumerator **SCROLL_Y_REVERSE**

enumerator **SCROLL_X_REVERSE**

enumerator **VERTICAL_LEFT**

enumerator **VERTICAL_RIGHT**

enum **FONT_SRC_TYPE**

font type enum

Values:

enumerator **GUI_FONT_SRC_BMP**

enumerator **GUI_FONT_SRC_STB**

enumerator **GUI_FONT_SRC_IMG**

enumerator **GUI_FONT_SRC_MAT**

enumerator **GUI_FONT_SRC_FT**

enumerator **GUI_FONT_SRC_TTF**

enum **FONT_SRC_MODE**

Values:

enumerator **FONT_SRC_MEMADDR**

enumerator **FONT_SRC_FILESYS**

enumerator **FONT_SRC_FTL**

Functions

void **gui_text_click**(*gui_text_t* *this_widget, gui_event_cb_t event_cb, void *parameter)

set textbox click event cb .

Parameters

- **this_widget** – text widget.
- **event_cb** – cb function.
- **parameter** – cb parameter.

void **gui_text_pswd_done**(*gui_text_t* *this_widget, gui_event_cb_t event_cb, void *parameter)

set textbox password done event cb, to handle password.

Parameters

- **this_widget** – text widget.
- **event_cb** – cb function.
- **parameter** – cb parameter.

void **gui_text_set**(*gui_text_t* *this_widget, void *text, *FONT_SRC_TYPE* text_type, gui_color_t color, uint16_t length, uint8_t font_size)

set the string in a text box widget.

Note: The font size must match the font file!

Parameters

- **this_widget** – the text box widget pointer.
- **text** – the text string.
- **text_type** – text type.
- **color** – the text's color.
- **length** – the text string's length.
- **font_size** – the text string's font size.

Returns
void

void **gui_text_set_animate**(void *o, uint32_t dur, int repeat_count, void *callback, void *p)

set animate.

Parameters

- **o** – text widget.
- **dur** – durtion. time length of the animate.
- **repeat_count** – 0:one shoot -1:endless.
- **callback** – happens at every frame.
- **p** – callback's parameter.

void **gui_text_mode_set**(*gui_text_t* *this_widget, *TEXT_MODE* mode)

set text mode of this_widget text widget.

Note: if text line count was more than one, it will display on the left even if it was set lft or right.

Parameters

- **this_widget** – the text widget pointer.
- **mode** – there was three mode: LEFT, CENTER and RIGHT.

void **gui_text_input_set**(*gui_text_t* *this_widget, bool inputable)

set inputable.

Parameters

- **this_widget** – the text box widget pointer.
- **inputable** – inputable.

void **gui_text_wordwrap_set**(*gui_text_t* *this_widget, bool wordwrap)

By setting wordwrap to enable English word wrapping.

Parameters

- **this_widget** – the text box widget pointer.
- **wordwrap** – wordwrap.

void **gui_text_use_matrix_by_img**(*gui_text_t* *this_widget, bool use_img_blit)

Enable/disable matrix-based image rendering for text.

Parameters

- **this** – Pointer to the text widget
- **use_img_blit** – true = use image matrix blitting (for complex transformations), false = use standard rendering

void **gui_text_rendermode_set**(*gui_text_t* *this_widget, uint8_t rendermode)

Set ttf raster render mode.

Parameters

- **this_widget** – the text box widget pointer.
- **rendermode** – rendermode.1/2/4/8

void **gui_text_set_min_scale**(*gui_text_t* *this_widget, float min_scale)

set text min scale.

Parameters

- **this** – the text box widget pointer.
- **min_scale** – minimum scale.

void **gui_text_move**(*gui_text_t* *this_widget, int16_t x, int16_t y)

move the text widget.

Parameters

- **this_widget** – the text box widget pointer.
- **x** – the X-axis coordinate of the text box.
- **y** – the Y-axis coordinate of the text box.

void **gui_text_size_set**(*gui_text_t* *this_widget, uint8_t height, uint8_t width)

set font size or width and height.

Note: if use freetype, width and height is effective, else height will be applied as font size.

Parameters

- **this_widget** – the text widget pointer.
- **height** – font height or font size.
- **width** – font width(only be effective when freetype was used).

void **gui_text_font_mode_set**(*gui_text_t* *this_widget, *FONT_SRC_MODE* font_mode)

set text font mode.

Parameters

- **this_widget** – the text widget pointer.
- **font_mode** – font source mode.

void **gui_text_type_set**(*gui_text_t* *this_widget, void *font_source, *FONT_SRC_MODE* font_mode)

set font type.

Note: The type must match the font size!

Parameters

- **this_widget** – the text widget pointer.
- **font_source** – the addr of .ttf or .bin.
- **font_mode** – font source mode.

void **gui_text_emoji_set**(*gui_text_t* *this_widget, uint8_t *path, uint8_t size)

Set emoji file path and emoji size.

Note: Need romfs.

Note: Example of a full emoji image file path: "font/emoji/emoji_u1f30d.bin".

Parameters

- **this** – The text widget pointer.
- **path** – Path contain folder path and file name prefix. Path eg:"font/emoji/emoji_u". Folder path is emoji image file folder path, eg:"font/emoji/". File name prefix is prefix before the filename for Unicode sorting, eg:"emoji_u".
- **size** – Emoji image file size. eg 32.

void **gui_text_encoding_set**(*gui_text_t* *this_widget, *TEXT_CHARSET* charset)

set font encoding.

Note: utf-8 or unicode.

Parameters

- **this_widget** – the text widget pointer.
- **encoding_type** – encoding_type.

void **gui_text_set_matrix**(*gui_text_t* *this_widget, gui_matrix_t *matrix)

set text matrix

Note:

Parameters

- **this_widget** – the text widget pointer.

- **encoding_type** – encoding_type.

void **gui_text_content_set**(*gui_text_t* *this_widget, void *text, uint16_t length)

set text content.

Parameters

- **this_widget** – the text widget pointer.
- **text** – the text string.
- **length** – the text string's length.

void **gui_text_convert_to_img**(*gui_text_t* *this_widget, GUI_FormatType font_img_type)

to draw text by img, so that text can be scaled.

Parameters

- **this_widget** – the text widget pointer.
- **font_img_type** – color format.

gui_text_t ***gui_text_create**(void *parent, const char *name, int16_t x, int16_t y, int16_t w, int16_t h)

create a text box widget.

Note: The area of the text box should be larger than that of the string to be shown, otherwise, part of the text will be hidden.

Parameters

- **parent** – the father widget which the text nested in.
- **filename** – the widget's name.
- **x** – the X-axis coordinate of the text box.
- **y** – the Y-axis coordinate of the text box.
- **w** – the width of the text box.
- **h** – the hight of the text box.

Returns

return the widget object pointer.

struct **gui_text_t**

text widget structure

Public Members

gui_obj_t **base**

gui_color_t **color**

gui_animate_t ***animate**

gui_img_t ***scale_img**

uint8_t ***emoji_path**

float **min_scale**

void ***content**

void ***data**

void ***path**

gui_matrix_t ***matrix**

uint16_t **len**

uint16_t **font_len**

uint16_t **active_font_len**

int16_t **char_width_sum**

int16_t **char_height_sum**

int16_t **char_line_sum**

int16_t **offset_x**

int16_t **offset_y**

TEXT_MODE **mode**

TEXT_CHARSET **charset**

FONT_SRC_TYPE **font_type**

FONT_SRC_MODE **font_mode**

uint8_t **font_height**

uint8_t **emoji_size**

uint8_t **checksum**

bool **layout_refresh**

bool **content_refresh**

bool **use_img_blit**

uint8_t **inputable**

uint8_t **ispasswd**

uint8_t **wordwrap**

uint8_t **scope**

uint8_t **rendermode**

struct **gui_text_line_t**

text line structure

Public Members

uint16_t **line_char**

uint16_t **line_dx**

3.4 3D Model

The widget supports loading 3D models composed of `.obj` and `.mtl` files, and supports adding animation effects.

3.4.1 GUI Load 3D Model

1. Components of a 3D model
   - `.obj` file: Stores the geometric data of the 3D model, including vertices, normals, texture coordinates, faces, etc.
   - `.mtl` file: Describes the material properties of the 3D model, including color, glossiness, transparency, and texture mapping.
   - Image files: Textures used in the model.

Fig. 1: Example of 3D Model Components

2. Parsing the 3D model and generating a 3D information descriptor
   - Invoke a script to process the `.obj` file.

Fig. 2: Script Processing

   - Generate a 3D information descriptor, which includes parsed OBJ data, parsed MTL data, and texture maps.

Fig. 3: Generating Binary Arrays

3. GUI load descriptor

   Place the desc file containing parsed obj data, mtl data, and image data into the project directory, and load it using *gui_3d_create()*.

   Example:

```
void *test_3d = gui_3d_create(gui_obj_get_root(), "3d-widget", (void *)_acdesc, 0,
↪ 0, 480, 480);
```

3.4.2 3D Widget Usage

Create Widget

Use *gui_3d_create()* to create the 3D model. The imported `desc_addr` file is the parsed data extracted by the script.

Global Shape Transformation

Use *gui_3d_set_global_shape_transform_cb()* to apply a global transformation to the 3D model, where `cb` sets the same shape transformation for all faces of the object. In this function, `world` and `camera` represent the world coordinate transformation of the 3D object and the camera view projection, respectively. Additionally, rectangular faces support the setting of `light` information.

Local Shape Transformation

Use *gui_3d_set_local_shape_transform_cb()* to apply a local transformation to the 3D model, where `cb` allows setting different shape transformations for each face of the object, and `face_index` specifies the face to be transformed. In this function, `world` and `camera` represent the world coordinate transformation of the 3D object and the camera view projection, respectively. Additionally, rectangular faces support the setting of `light` information.

World Transformation

The initialization function is `gui_3d_world_inititalize(gui_3d_matrix_t *world, float x, float y, float z, float rotX, float rotY, float rotZ, float scale)`.

- `world`: A pointer to the world transformation matrix, it transforms the 3D object from model coordinates to world coordinates.
- `x`: The distance of translation along the X-axis, used to determine the object's position in the X direction within the world coordinate system.
- `y`: The distance of translation along the Y-axis, used to determine the object's position in the Y direction within the world coordinate system.
- `z`: The distance of translation along the Z-axis, used to determine the object's position in the Z direction within the world coordinate system.
- `rotX`: The angle of rotation around the X-axis (in degrees).
- `rotY`: The angle of rotation around the Y-axis (in degrees).
- `rotZ`: The angle of rotation around the Z-axis (in degrees).
- `scale`: A uniform scaling factor used to proportionally scale the object in all directions.

Purpose:

1. The world transformation matrix typically handles transforming the model coordinate system to the world coordinate system. For example, if an object is located at the origin of the model coordinate system, it can be moved to any position in the scene and scaled/rotated through world transformation.
2. Performing independent world transformations for each face can achieve localized animations or static displays.
3. Different faces can share the same world matrix, or you can use `gui_3d_calculator_matrix(gui_3d_matrix_t *matrix, float x, float y, float z, gui_point_4d_t point, gui_vector_4d_t vector, float degrees, float scale)` to generate different matrices for each face to achieve personalized local transformations.

Camera Transformation

The initialization function is `gui_3d_camera_UVN_initialize(gui_3d_camera_t *camera, gui_point_4d_t cameraPosition, gui_point_4d_t cameraTarget, float near, float far, float fov, float viewPortWidth, float viewPortHeight)`.

- `camera`: A pointer to the camera structure, used to initialize camera properties.
- `cameraPosition`: The position of the camera in world coordinates.
- `cameraTarget`: The target point the camera is directed at, i.e., the focal point of the camera's line of sight.
- `near`: The near clipping plane distance, defining the distance from the camera to the near plane of the camera's view frustum. Objects closer than this distance will be clipped.
- `far`: The far clipping plane distance, defining the distance from the camera to the far plane of the view frustum. Objects farther than this distance will be clipped.
- `fov`: The field of view, usually expressed as a vertical angle (in degrees), defining the openness of the camera, i.e., the opening angle of the camera's view frustum.
- `viewPortWidth`: The width of the viewport, defining the horizontal size of the rendering target or window.
- `viewPortHeight`: The height of the viewport, defining the vertical size of the rendering target or window.

Purpose:

1. Camera transformation defines the observer's position and direction in the scene, transforming the world coordinate system to the camera coordinate system.
2. By manipulating the camera, different perspectives can be achieved, such as translating the camera position or changing the viewing direction.

Lighting Information

The initialization function is `gui_3d_light_inititalize(gui_3d_light_t *light, gui_point_4d_t lightPosition, gui_point_4d_t lightTarget, float included_angle, float blend_ratio, gui_3d_RGBAcolor_t color)`.

- `light`: A pointer to the light source structure, used to initialize the properties of the light source.
- `lightPosition`: The position of the light source in world coordinates.
- `lightTarget`: The target position of the light source, defining the direction of illumination.
- `included_angle`: The cone angle of the light (in degrees), represented as angle α in the diagram. It determines the illumination range of the spotlight, which corresponds to the outer circle of the spotlight in the diagram.
- `blend_ratio`: The ratio of the light blending region, defining the softness of the spotlight's edge. It ranges from 0 to 1 and determines angle β in the diagram. The value is calculated using the following formula:

$$= (1 - ratio)$$

  The blending region extends from the inner circle to the outer circle of the spotlight. Within the inner circle, the light intensity is constant, while it gradually diminishes from the inner to the outer circle.

- `color`: The color of the light source and its transparency.

Fig. 4: Example of Spotlight Effect

Purpose:

1. The light source type is a spotlight, and its properties include initial position, light direction, cone angle, blend ratio, and light color.
2. Adjusting lighting locally for each face or object can create different visual styles.

Set Animation

The *`gui_obj_create_timer()`* function can be used to set animation properties for a 3D object. The `callback` parameter is a callback function for animation updates.

3.4.3 Example

3D Butterfly

The model is composed entirely of rectangular faces. By calling *gui_3d_set_local_shape_transform_cb()*, you can set local transformations for different faces to create animation effects.

```
#include "guidef.h"
#include "gui_img.h"
#include "gui_obj.h"
#include "string.h"
#include "stdio.h"
#include "stdlib.h"
#include "gui_server.h"
#include "gui_components_init.h"
#include "gui_canvas.h"
#include "gui_3d.h"

#include "butterfly/desc.txt"
#include "math.h"
#include "tp_algo.h"

static int frame_counter = 0;
static float wing_angle = 0.0f;
static float butterfly_x = 0.0f;
static float butterfly_y = 0.0f;
static float butterfly_z = 0.0f;
static float butterfly_rz = 0.0f;

bool is_moving_to_target = false;
static float target_dx = 0.0f;
static float target_dy = 0.0f;
static float source_dx = 0.0f;
static float source_dy = 0.0f;
static float move_speed = 0.02f;
static float wing_time = 0.0f;
void update_animation()
{
    touch_info_t *tp = tp_get_info();
    gui_dispdev_t *dc = gui_get_dc();

    if (tp->pressed)
    {
        target_dx = (tp->x - dc->screen_width / 2) / 2.5f;
        target_dy = (tp->y - dc->screen_height / 2) / 2.5f;
        is_moving_to_target = true;
    }

    if (is_moving_to_target)
    {
        float dx = target_dx - source_dx;
        float dy = target_dy - source_dy;

        float distance = sqrtf(dx * dx + dy * dy);

        if (distance > 10.0f)
        {
```

(continues on next page)

(continued from previous page)

```
            // Acceleration and deceleration
            float speed_factor = fminf(distance / 40.0f, 1.0f);
            source_dx += dx * move_speed * speed_factor;
            source_dy += dy * move_speed * speed_factor;

            // Caculate new rotate angle
            float desired_angle = atan2f(dy, dx) * (180.0f / M_PI) + 90;
            float angle_difference = desired_angle - butterfly_rz;

            if (angle_difference > 180.0f)
            {
                angle_difference -= 360.0f;
            }
            if (angle_difference < -180.0f)
            {
                angle_difference += 360.0f;
            }
            butterfly_rz += angle_difference * 0.1f;

            // Adjust wing flapping frequency based on speed
            wing_time += 0.2f + speed_factor * 0.2f;
            wing_angle = 60.0f * sinf(wing_time);

            butterfly_x = -source_dx;
            butterfly_y = -source_dy;
        }
        else
        {
            is_moving_to_target = false;
        }
    }
    else
    {
        frame_counter++;
        wing_time += 0.1f;
        wing_angle = 50.0f * sinf(wing_time);
        butterfly_z = 5.0f * sinf(frame_counter * 0.05f);
    }

}

static void cb(void *this, size_t face_index/*face offset*/, gui_3d_world_t *world,
               gui_3d_camera_t *camera, gui_3d_light_t *light)
{
    gui_dispdev_t *dc = gui_get_dc();
    gui_3d_matrix_t face_matrix;
    gui_3d_matrix_t object_matrix;

    gui_3d_camera_UVN_initialize(camera, gui_point_4d(0, 0, 80), gui_point_4d(0, 0,␣
↪0), 1, 32767, 90,
                                 dc->screen_width, dc->screen_height);

    gui_3d_world_inititalize(&object_matrix, butterfly_x, butterfly_y, butterfly_z, 0,
↪ 0,
                             butterfly_rz,
                             5);
```

(continues on next page)

(continued from previous page)

```
    if (face_index == 0)
    {
        gui_3d_calculator_matrix(&face_matrix, 0, 0, 0, gui_3d_point(0, 0, 0), gui_3d_
↪vector(0, 1, 0),
                                 wing_angle, 1);
    }
    else if (face_index == 1)
    {
        gui_3d_calculator_matrix(&face_matrix, 0, 0, 0, gui_3d_point(0, 0, 0), gui_3d_
↪vector(0, 1, 0),
                                 -wing_angle, 1);
    }
    else if (face_index == 2)
    {
        gui_3d_calculator_matrix(&face_matrix, 0, 0, 0, gui_3d_point(0, 0, 0), gui_3d_
↪vector(0, 1, 0),
                                 wing_angle, 1);
    }
    else if (face_index == 3)
    {
        gui_3d_calculator_matrix(&face_matrix, 0, 0, 0, gui_3d_point(0, 0, 0), gui_3d_
↪vector(0, 1, 0),
                                 -wing_angle, 1);
    }
    else
    {
        gui_3d_calculator_matrix(&face_matrix, 0, 0, 0, gui_3d_point(0, 0, 0), gui_3d_
↪vector(0, 1, 0), 0,
                                 1);
    }

    *world = gui_3d_matrix_multiply(face_matrix, object_matrix);

}



static int app_init(void)
{
    void *test_3d = gui_3d_create(gui_obj_get_root(), "3d-widget", (void *)_acdesc, 0,
↪ 0, 480, 480);

    gui_3d_set_local_shape_transform_cb(test_3d, 0, (gui_3d_shape_transform_cb)cb);

    gui_obj_create_timer(&(((gui_3d_base_t *)test_3d)->base), 17, true, update_
↪animation);
    gui_obj_start_timer(&(((gui_3d_base_t *)test_3d)->base));

    return 0;

}
```

3D Prism

The model is composed entirely of rectangular faces. By calling `gui_3d_light_inititalize()`, you can add lighting effects.

```
#include "math.h"
#include "cube3D/desc.txt"

static float rot_angle = 0.0f;
void update_cube_animation()
{
   rot_angle++;
}

static void cube_cb(gui_3d_t *this, size_t face/*face offset*/, gui_3d_world_t *world,
               gui_3d_camera_t *camera, gui_3d_light_t *light)
{
   gui_dispdev_t *dc = gui_get_dc();
   gui_3d_matrix_t face_matrix;
   gui_3d_matrix_t object_matrix;

   gui_3d_camera_UVN_initialize(camera, gui_point_4d(0, 6, 15), gui_point_4d(0, 0, 0),
↪ 1, 32767, 90,
                                dc->screen_width, dc->screen_height);

   gui_3d_world_inititalize(&object_matrix, 0, 22, 40, 90, 0, 0,
                            10);

   gui_3d_light_inititalize(light, gui_point_4d(0, 22, 45), gui_point_4d(0, 22, 40),␣
↪60, 0.6, (gui_3d_RGBAcolor_t){255, 215, 0, 255});

   gui_3d_calculator_matrix(&face_matrix, 0, 0, 0, gui_3d_point(0, 0, 0), gui_3d_
↪vector(0, 0, 1), rot_angle,
                                1);

   *world = gui_3d_matrix_multiply(face_matrix, object_matrix);

}
static int app_init(void)
{
   void *test_3d = gui_3d_create(gui_obj_get_root(), "3d-widget", (void *)_acdesc, 0,␣
↪0, 480, 480);

   gui_3d_set_global_shape_transform_cb(test_3d, (gui_3d_shape_transform_cb)cube_cb);

   gui_obj_create_timer(&(((gui_3d_base_t *)test_3d)->base), 17, true, update_cube_
↪animation);
   gui_obj_start_timer(&(((gui_3d_base_t *)test_3d)->base));

   return 0;
}
```

3D Face

The model is composed of 1,454 triangular faces.

```
#include "guidef.h"
#include "gui_img.h"
#include "gui_obj.h"
#include "string.h"
#include "stdio.h"
#include "stdlib.h"
#include "gui_server.h"
#include "gui_components_init.h"

#include "gui_3d.h"
#include "tp_algo.h"
#include "face3d/desc_1454.txt"
#include "face3d/desc_5822.txt"

static float rot_angle = 0.0f;

void update_face_animation()
{
    touch_info_t *tp = tp_get_info();

    if (tp->pressed || tp->pressing)
    {
        rot_angle += tp->deltaX / 5.0f;
    }
}


static void face_cb(void *this, gui_3d_world_t *world,
                    gui_3d_camera_t *camera)
{
    gui_dispdev_t *dc = gui_get_dc();
    gui_3d_matrix_t face_matrix;
    gui_3d_matrix_t object_matrix;

    gui_3d_camera_UVN_initialize(camera, gui_point_4d(0, 3, 60), gui_point_4d(0, 0,␣
↪0), 1, 32767, 90,
                                 dc->screen_width, dc->screen_height);

    // gui_3d_world_inititalize(&object_matrix, 0, 25, 120, 0, 0, 0,
    //                          5);

    // gui_3d_calculator_matrix(&face_matrix, 0, 0, 0, gui_3d_point(0, 0, 0), gui_3d_
↪vector(0, 1, 0),
    //                          rot_angle,
    //                          1);

    // *world = gui_3d_matrix_multiply(face_matrix, object_matrix);

    gui_3d_world_inititalize(world, 0, 25, 120, 0, rot_angle, 0, 5);
}



static int app_init(void)
```

(continues on next page)

(continued from previous page)

```
{
    void *test_3d = gui_3d_create(gui_obj_get_root(), "3d-widget", (void *)_acdesc_
↪1454, 0, 0, 480,
                                  480);
    gui_3d_set_global_shape_transform_cb(test_3d, (gui_3d_shape_transform_cb)face_cb);

//    extern void gui_fps_create(void *parent);
//    gui_fps_create(&(app->screen));

    gui_obj_create_timer(&(((gui_3d_base_t *)test_3d)->base), 17, true, update_face_
↪animation);
    gui_obj_start_timer(&(((gui_3d_base_t *)test_3d)->base));

    return 0;

}
```

3.4.4 API

Typedefs

typedef void (*__gui_3d_shape_transform_cb__)(void *this, gui_3d_world_t *world, gui_3d_camera_t *camera, void *extra)

Functions

void *__gui_3d_create__(void *parent, const char *name, void *desc_addr, int16_t x, int16_t y, int16_t w, int16_t h)

3d widget create

Parameters

- **parent** – parent widget
- **name** – widget name
- **desc_addr** – description file data
- **x** – the X-axis coordinate relative to parent widget
- **y** – the Y-axis coordinate relative to parent widget
- **w** – width
- **h** – height

Returns

the widget object pointer

void __gui_3d_set_global_shape_transform_cb__(void *this, *gui_3d_shape_transform_cb* cb)

set global shape transform callback

Parameters

- **this** – the 3d widget pointer

- **cb** – Set callback functions for the world coordinate system, camera coordinate system, and light source for all faces

void **gui_3d_set_local_shape_transform_cb**(void *this, size_t face, *gui_3d_shape_transform_cb* cb)

set local shape transform callback

Parameters

- **this** – the 3d widget pointer
- **face** – face offset
- **cb** – Set callback functions for the world coordinate system, camera coordinate system, and light source for the specified face

void **gui_3d_on_click**(void *this, void *callback, void *parameter)

Set a callback function for when the 3D widget is clicked.

Parameters

- **this** – Pointer to the 3D widget.
- **callback** – Callback function to execute on click.
- **parameter** – Additional parameter for the callback.

struct **gui_3d_base_t**

Public Members

gui_obj_t **base**

gui_3d_description_t ***desc**

3.5 View

The view widget is a kind of container that makes switching more convenient. Any new view widget can be created in real time in response to an event(clicking and sliding in all four directions…) and multiple switching effects can be selected. During the switching process, there will be two views in the memory, and after the switching is completed, the undisplayed view will be automatically cleaned up, which can effectively reduce the memory consumption.

3.5.1 Usage

Register Descriptor of View

The *gui_view_descriptor_register()* function can be used to register descriptor of view in the descriptor list for other view to read and use as a parameter to create the view, via passing in the descriptor's address. The `gui_view_descriptor` structure is defined as follows:

```
typedef struct gui_view_descriptor
{
    const char *name;
    gui_view_t **pView;

    void (* on_switch_in)(gui_view_t *view);  // callback function when view is␣
↪switched in and created
    void (* on_switch_out)(gui_view_t
                           *view); // callback function when view is switched out and␣
↪destroyed

    uint8_t keep       : 1;
} gui_view_descriptor_t; // if keep is true, the view will not be destroyed when␣
↪switch to other view and will be created when register view
```

Get Descriptor of View by Name

The *gui_view_descriptor_get()* function can be used to get the view descriptor with the corresponding `name` by passing in the string.

Create View Widget

The *gui_view_create()* function can be used to establish a view widget.

Set Switch View Event

The *gui_view_switch_on_event()* function can be used to set switch view event. Repeatable settings for a particular event will use the latest descriptor. Specific events include `GUI_EVENT_TOUCH_CLICKED` `GUI_EVENT_KB_SHORT_CLICKED` `GUI_EVENT_TOUCH_MOVE_LEFT` `GUI_EVENT_TOUCH_MOVE_RIGHT` and so on. The available switching styles include the following:

```
typedef enum
{
    VIEW_STILL            = 0x0000, ///< Overlay effect with new view transplate in
    VIEW_TRANSPLATION     = 0x0001, ///< Transplate from the slide direction
    VIEW_REDUCTION        = 0x0002, ///< Zoom in from the slide direction
    VIEW_ROTATE           = 0x0003, ///< Rotate in from the slide direction
    VIEW_CUBE             = 0x0004, ///< Rotate in from the slide direction like cube
    VIEW_ANIMATION_NULL   = 0x0005,
    VIEW_ANIMATION_1,               ///< Recommended for startup
    VIEW_ANIMATION_2,               ///< Recommended for startup
    VIEW_ANIMATION_3,               ///< Recommended for startup
    VIEW_ANIMATION_4,               ///< Recommended for startup
    VIEW_ANIMATION_5,               ///< Recommended for startup
    VIEW_ANIMATION_6,               ///< Recommended for shutdown
    VIEW_ANIMATION_7,               ///< Recommended for shutdown
    VIEW_ANIMATION_8,               ///< Recommended for shutdown
} VIEW_SWITCH_STYLE;
```

Switch View Directly

The *gui_view_switch_direct()* function can be used to switch view directly, which can be used in conjunction with events or animations of the child widgets based on view. Note that the switching style is limited to the animation style and cannot be set to the sliding style.

Get Current View Pointer

The *gui_view_get_current_view()* function can be used to get current view pointer, and can be used with *gui_view_switch_direct()* to switch the current view.

3.5.2 Example

View

Below are three separate C files, each containing a descriptor for the view and the design function.

3.5.3 API

Defines

EVENT_NUM_MAX

Enums

enum **VIEW_SWITCH_STYLE**

Values:

enumerator **VIEW_STILL**

Overlay effect with new view transplate in.

enumerator **VIEW_TRANSPLATION**

Transplate from the slide direction.

enumerator **VIEW_REDUCTION**

Zoom in from the slide direction.

enumerator **VIEW_ROTATE**

Rotate in from the slide direction.

enumerator **VIEW_CUBE**

Rotate in from the slide direction like cube.

enumerator **VIEW_ANIMATION_NULL**

enumerator **VIEW_ANIMATION_1**

Recommended for startup.

enumerator **VIEW_ANIMATION_2**

Recommended for startup.

enumerator **VIEW_ANIMATION_3**

Recommended for startup.

enumerator **VIEW_ANIMATION_4**

Recommended for startup.

enumerator **VIEW_ANIMATION_5**

Recommended for startup.

enumerator **VIEW_ANIMATION_6**

Recommended for shutdown.

enumerator **VIEW_ANIMATION_7**

Recommended for shutdown.

enumerator **VIEW_ANIMATION_8**

Recommended for shutdown.

Functions

gui_view_t ***gui_view_create**(void *parent, const *gui_view_descriptor_t* *descriptor, int16_t x, int16_t y, int16_t w, int16_t h)

Create a view widget.

Parameters

- **parent** – The father widget it nested in.
- **descriptor** – Pointer to a descriptor that defines the new view to switch to.
- **x** – The X-axis coordinate relative to parent widget
- **y** – The Y-axis coordinate relative to parent widget
- **w** – Width
- **h** – Height

Returns

return the widget object pointer.

void **gui_view_descriptor_register**(const *gui_view_descriptor_t* *descriptor)

Register view's descriptor.

Parameters

descriptor – Pointer to a descriptor that defines the new view to switch to.

const *gui_view_descriptor_t* ***gui_view_descriptor_get**(const char *name)

Get target view's descriptor by name.

Parameters

name – View descriptor's name that can used to find target view.

void **gui_view_switch_on_event**(*gui_view_t* *_this, const *gui_view_descriptor_t* *descriptor, *VIEW_SWITCH_STYLE* switch_out_style, *VIEW_SWITCH_STYLE* switch_in_style, gui_event_t event)

Switches the current GUI view to a new view based on the specified event.

This function handles the transition between GUI views. It takes the current view context and switches it to a new view as described by the `descriptor`. The transition is triggered by a specified event and can be customized with different switch styles for the outgoing and incoming views.

Parameters

- **_this** – Pointer to the current GUI view context that is being manipulated.
- **descriptor** – Pointer to a descriptor that defines the new view to switch to.
- **switch_out_style** – Style applied to the outgoing view during the switch.
- **switch_in_style** – Style applied to the incoming view during the switch.
- **event** – The event that triggers the view switch.

void **gui_view_switch_direct**(*gui_view_t* *_this, const *gui_view_descriptor_t* *descriptor, *VIEW_SWITCH_STYLE* switch_out_style, *VIEW_SWITCH_STYLE* switch_in_style)

Switches directly the current GUI view to a new view through animation.

This function handles the transition between GUI views. It takes the current view context and switches it to a new view as described by the `descriptor`. The transition animation can be customized with different animation switch styles for the outgoing and incoming views.

Parameters

- **_this** – Pointer to the current GUI view context that is being manipulated.
- **descriptor** – Pointer to a descriptor that defines the new view to switch to.
- **switch_out_style** – Style applied to the outgoing view during the switch.
- **switch_in_style** – Style applied to the incoming view during the switch.

gui_view_t ***gui_view_get_current_view**(void)

Get current view pointer.

Returns

return current view pointer.

struct **gui_view_id_t**

Public Members

int8_t **x**

int8_t **y**

struct **gui_view_t**

Public Members

gui_obj_t **base**

int16_t **release_x**

int16_t **release_y**

gui_animate_t ***animate**

gui_view_id_t **cur_id**

VIEW_SWITCH_STYLE **style**

const struct gui_view_descriptor ***descriptor**

uint32_t **view_switch_ready**

uint32_t **event**

uint32_t **moveback**

uint32_t **view_tp**

uint32_t **view_left**

uint32_t **view_right**

uint32_t **view_up**

uint32_t **view_down**

uint32_t **view_click**

uint32_t **view_touch_long**

uint32_t **view_button**

uint32_t **view_button_long**

struct gui_view_on_event **__on_event__

uint8_t **on_event_num**

uint8_t **checksum**

struct **gui_view_descriptor_t**

Public Members

const char *__name__

gui_view_t **__pView__

void (*__on_switch_in__)(*gui_view_t* *view)

void (*__on_switch_out__)(*gui_view_t* *view)

uint8_t **keep**

struct **gui_view_on_event_t**

Public Members

const *gui_view_descriptor_t* *__descriptor__

VIEW_SWITCH_STYLE **switch_out_style**

VIEW_SWITCH_STYLE **switch_in_style**

gui_event_t **event**

CHAPTER

FOUR

PORTING

Porting consists of two parts: platform porting and display scheme extension. The display scheme extension currently supports font library porting.

4.1 Platform Porting

The porting files are located in the `gui_port` folder. Six files need to be modified, with their filenames and functions as follows.

Filename	Description
`gui_port_acc.c`	Acceleration
`gui_port_dc.c`	Display Device
`gui_port_filesystem.c`	Filesystem
`gui_port_ftl.c`	Flash Translation Layer
`gui_port_indev.c`	Input Device
`gui_port_os.c`	Operating System

Currently, porting has been done on FreeRTOS, RT-Thread, and Windows for reference.

4.1.1 Acceleration

- Refer to `guidef.h` and `gui_port_acc.c`.
- Define the accelerated drawing interface depending on the platform model, generally `hw_acc_blit` or `sw_acc_blit`.
- The structure definition is as follows:

```
typedef struct acc_engine
{
    void (*blit)(draw_img_t *image, gui_dispdev_t *dc, gui_rect_t *rect);
} acc_engine_t;
```

4.1.2 Display Device

- Refer to `guidef.h` and `gui_port_dc.c`.
- Define the screen width and height, framebuffer address and mode, whether the resolution is scaled, and implement the refresh function. Refer to `guidef.h` for the structure definition.
- A typical `gui_dispdev` structure initialization declaration is as follows:

```
static struct gui_dispdev dc =
{
    .bit_depth = DRV_PIXEL_BITS,
    .fb_width = DRV_LCD_WIDTH,
    .fb_height = FB_HEIGHT,
    .screen_width =  DRV_LCD_WIDTH,
    .screen_height = DRV_LCD_HIGHT,
    .dc.disp_buf_1 = disp_write_buff1_port,
    .dc.disp_buf_2 = disp_write_buff2_port,
    .driver_ic_fps = 60,
    .driver_ic_hfp = 10,
    .driver_ic_hbp = 10,
    .driver_ic_active_width = DRV_LCD_WIDTH,
    .type = DC_RAMLESS,
    .adaption = false,
    .section = {0, 0, 0, 0},
    .section_count = 0,
    .lcd_update = port_gui_lcd_update,
    .flash_seq_trans_disable = flash_boost_disable,
    .flash_seq_trans_enable = flash_boost_enable,
    .reset_lcd_timer = reset_vendor_counter,
    .get_lcd_us = read_vendor_counter_no_display,
    .lcd_te_wait = port_lcd_te_wait,
    .dc.scale_x = 1,
    .dc.scale_y = 1,
};
```

- In `DC_SINGLE` mode, the framebuffer size is `screen_width * screen_height * bit_depth / 8`.
- In `DC_RAMLESS` mode, two partial framebuffers are used, with size `fb_width * fb_height * bit_depth / 8`, where `fb_height` is the segmented height.

Interface

The following table lists the LCD-related interfaces supported by mainstream chips. If you want to know more information, please click on the specific chip name.

SOC	8080	QSPI	RGB	MIPI	SPI
RTL8762C	Y	NA	NA	NA	Y
RTL8762D	Y	Y	NA	NA	Y
RTL8763E	Y	Y	NA	NA	Y
RTL8772G	Y	Y	Y	NA	Y
RTL8773E	Y	Y	Y	NA	Y

Note: 'Y' means the driver is already included in the library. 'NA' means the driver is not yet included in the library.

Driver IC

The following table lists the LCD-related driver ICs supported by mainstream chips. If you want to know more information, please click on the specific chip name.

SOC	EK97	ICNA3	NT35	NV30	ST77(	ST77	ST77	OTM80	SH86(	SH86(	RM69	ST77	NV3041A
RTL87	NA	NA	NA	NA	NA	NA	Y	NA	NA	NA	Y	Y	Y
RTL87	NA	NA	Y	NA	NA	NA	NA	NA	NA	Y	NA	NA	NA
RTL87	Y	Y	Y	Y	Y	Y	Y	NA	NA	NA	NA	NA	NA
RTL87	NA	NA	NA	NA	NA	NA	NA	NA	Y	NA	NA	NA	NA

Note: 'Y' means the driver is already included in the library. 'NA' means the driver is not yet included in the library.

4.1.3 Filesystem

- Refer to `guidef.h` and `gui_port_filesystem.c`
- Define several posix-like interfaces to operate files and directories.
- If not using a filesystem, you can fill in null pointers.
- The structure definition is as follows:

```
struct gui_fs
{
    int (*open)(const char *file, int flags, ...);
    int (*close)(int d);
    int (*read)(int fd, void *buf, size_t len);
    int (*write)(int fd, const void *buf, size_t len);
    int (*lseek)(int fd, int offset, int whence);
    /* directory api*/
    gui_fs_dir *(*opendir)(const char *name);
    struct gui_fs_dirent *(*readdir)(gui_fs_dir *d);
    int (*closedir)(gui_fs_dir *d);
    int (*ioctl)(int fildes, int cmd, ...);
    void (*fstat)(int fildes, gui_fs_stat_t *buf);
};
```

4.1.4 Flash Translation Layer

- Refer to `guidef.h` and `gui_port_ftl.c`
- Define three interfaces for the Flash Translation Layer: `read`, `write`, `erase`.
- If not using a Flash Translation Layer, you can fill in null pointers.
- The structure definition is as follows:

```
struct gui_ftl
{
    int (*read)(uint32_t addr, uint8_t *buf, uint32_t len);
    int (*write)(uint32_t addr, const uint8_t *buf, uint32_t len);
```

(continues on next page)

(continued from previous page)

```
    int (*erase)(uint32_t addr, uint32_t len);
};
```

4.1.5 Input Device

- Refer to `guidef.h` and `gui_port_indev.c`
- Input devices include touchpads, keyboards, and wheels. The structure for input information is as follows:

```
typedef struct gui_indev
{
    uint16_t tp_witdh;
    uint16_t tp_height;
    uint32_t touch_timeout_ms;
    uint16_t long_button_time_ms;
    uint16_t short_button_time_ms;
    uint16_t kb_long_button_time_ms;
    uint16_t kb_short_button_time_ms;
    uint16_t quick_slide_time_ms;

    void (*ext_button_indicate)(void (*callback)(void));

    gui_touch_port_data_t *(*tp_get_data)(void);

    gui_kb_port_data_t *(*kb_get_port_data)(void);

    gui_wheel_port_data_t *(*wheel_get_port_data)(void);
} gui_indev_t;
```

- If a specific input device is needed, the corresponding data acquisition function needs to be implemented in `gui_indev`, and the required time thresholds need to be filled in.

Touch IC

The following table lists the Touch-related ICs supported by all chips. If you want to know more information, please click on the specific chip name.

SOC	CST816S	CHSC6417	FT3169	GT911	ZT2717	CST816T	GT9147
RTL8762D	Y	NA	NA	NA	NA	NA	NA
RTL8763E	NA	NA	NA	NA	NA	Y	Y
RTL8772G	NA	NA	NA	Y	Y	NA	NA
RTL8773E	Y	NA	NA	Y	NA	NA	NA

Note: 'Y' means the driver is already included in the library. 'NA' means the driver is not yet included in the library.

4.1.6 Operating System

- Refer to `guidef.h` and `gui_port_os.c`
- Define the interfaces for thread, timer, message queue, and memory management. The structure definition is as follows:

```
typedef struct gui_os_api
{
    char *name;
    void *(*thread_create)(const char *name, void (*entry)(void *param), void␣
↪*param,
                           uint32_t stack_size, uint8_t priority);
    bool (*thread_delete)(void *handle);
    bool (*thread_suspend)(void *handle);
    bool (*thread_resume)(void *handle);
    bool (*thread_mdelay)(uint32_t ms);
    uint32_t (*thread_ms_get)(void);
    uint32_t (*thread_us_get)(void);
    bool (*mq_create)(void *handle, const char *name, uint32_t msg_size, uint32_t␣
↪max_msgs);
    bool (*mq_send)(void *handle, void *buffer, uint32_t size, uint32_t timeout);
    bool (*mq_send_urgent)(void *handle, void *buffer, uint32_t size, uint32_t␣
↪timeout);
    bool (*mq_recv)(void *handle, void *buffer, uint32_t size, uint32_t timeout);

    void *(*f_malloc)(uint32_t);
    void *(*f_realloc)(void *ptr, uint32_t);
    void (*f_free)(void *rmem);

    void (*gui_sleep_cb)(void);

    void *mem_addr;
    uint32_t mem_size;

    uint32_t mem_threshold_size;
    void *lower_mem_addr;
    uint32_t lower_mem_size;

    log_func_t log;
    void (*gui_tick_hook)(void);
} gui_os_api_t;
```

4.1.7 Sleep Management

To reduce power consumption and increase the device's usage time, sleep (low power) mode is supported.

- Refer to `gui_app.h`

```
typedef struct gui_app gui_app_t;
struct gui_app
{
    gui_obj_t screen;               //!< Root node of the control tree
    const char *xml;                //!< Control tree design file
    uint32_t active_ms;             //!< Screen off delay
    void *thread_id;                //!< Thread handle (optional)
    void (* thread_entry)(void *this); //!< Thread entry function
```

(continues on next page)

(continued from previous page)

```
    void (* ctor)(void *this);      //!< Constructor
    void (* dtor)(void *this);      //!< Destructor
    void (* ui_design)(gui_app_t *); //!< UI creation entry function
    bool lvgl;
    bool arm2d;
    bool close;
    bool next;
    bool close_sync;
};
```

`active_ms` is the standby time of the GUI application, which can be defined as different values in different applications. Like other types of electronic devices, when the screen continuously displays an interface for the standby time, the device will enter sleep mode. In sleep mode, the device can be awakened by touching the touchpad, pressing a key, or sending a message. In the chip manual, this low power state where peripherals can be turned off is called Deep Low Power State (DLPS). More information about DLPS can be found in the relevant SDK documentation.

4.2 Font Porting

This chapter will analyze the font library code segment and explain how to replace HoneyGUI's native font library with a custom one provided by the developer, or how to add customized features.

4.2.1 Dot Matrix Font Library Porting

Glyph Loading

Text Encoding Conversion

In the file `font_mem.c`, within the function *gui_font_get_dot_info()*, *process_content_by_charset()* parses the text content of the text widget and saves it as Unicode (UTF-32) in `unicode_buf`. The number of Unicode characters is returned in `unicode_len`.

```
uint32_t *unicode_buf = NULL;
uint16_t unicode_len = 0;
unicode_len = process_content_by_charset(text->charset, text->content, text->len, &
↪unicode_buf);
if (unicode_len == 0)
{
    gui_log("Warning! After process, unicode len of text: %s is 0!\n", text->base.
↪name);
    text->font_len = 0;
    return;
}
```

For the specific implementation of *process_content_by_charset()*, please refer to `draw_font.c`.

Note: The parsing process supports UTF-8, UTF-16, and UTF-32.

Subsequently, Unicode information in `unicode_buf` will be used to index text data from the font library.

Text encoding conversion for minor languages, such as Arabic character concatenation and other calculations involving Unicode, can be performed either before or after the encoding conversion. If the conversion is done later, `unicode_len` must be updated accordingly.

Note: The unit of `unicode_len` is bytes, not the number of characters.

Font Library Indexing

In the file `font_mem.c`, within the function *`gui_font_get_dot_info()`*, the Unicode value is parsed and then used to index glyph information from the font library designated by the text widget.

Since the font library tool has the `crop` attribute and two indexing modes, different parsing code is used to find text data and dot matrix data in the font library file using the Unicode value.

The purpose of the font library parsing code is to populate the `chr` structure array, which is structured as follows:

```
typedef struct
{
    uint32_t unicode;
    int16_t x;
    int16_t y;
    int16_t w;
    int16_t h;
    uint8_t char_y;
    uint8_t char_w;
    uint8_t char_h;
    uint8_t *dot_addr;
    uint8_t *buf;
    gui_img_t *emoji_img;
} mem_char_t;
```

Each member has the following meanings:

- `Unicode`: The Unicode of the dot matrix text, expressed in UTF-32LE format.
- `x`: The X-coordinate of the upper-left corner of the dot matrix text boundary, determined during layout, used to set the drawing coordinates of the text.
- `y`: The Y-coordinate of the upper-left corner of the dot matrix text boundary, determined during layout, used to set the drawing coordinates of the text.
- `w`: The data width of the character in the dot matrix data. Due to byte alignment and compression characteristics, this value is not always equal to the font size.
- `h`: The height of the dot matrix text, which is always equal to the font size, used to define the basic drawing area and for multi-line layout.
- `char_y`: The number of blank rows above the character, representing the Y-coordinate distance between the topmost pixel of the text dot matrix and the upper boundary, used to constrain the drawing area.
- `char_w`: The pixel width of the character, representing the difference in the X-coordinate between the leftmost boundary (starting point) and the rightmost pixel of the text. This value is used to constrain the drawing area during drawing and represents the text width during layout.
- `char_h`: The pixel height of the character, representing the Y-coordinate distance between the bottommost pixel of the text dot matrix and the upper boundary. The value of `char_h` minus `char_y` gives the actual pixel height of the dot matrix.

- `dot_addr`: The starting address of the dot matrix data corresponding to the text.
- `emoji_img`: The pointer to the widget corresponding to the Emoji image. This value is NULL if the Emoji feature is not used.

Fig. 1: Glyph Example

During the font library indexing phase, all members of `chr` except for the `x` and `y` coordinates will be populated to prepare for the next step of layout.

Note: Due to differences in data storage rules under different modes, the drawing areas also vary. For example, `char_y` and `char_h` are only effective when `crop=1` and `index_method=0`.

Since this stage involves using the Unicode to look up width information for the dot matrix text and the dot matrix data pointer, it's best to complete the Unicode-level text transformations before this step. For example, Arabic script ligatures should be handled in this stage, whereas Thai glyph fusion should be handled during the layout stage.

If you are porting using your custom font library, you can populate the `chr` data structures using information from your custom font library. The default parts can be used for the subsequent layout and drawing stages.

Layout

The text widget supports various layout modes.

The specific layout functionality is located in the file `font_mem.c` in the function *gui_font_mem_layout()*. Each layout mode has a different layout logic; however, all depend on the glyph information `chr` and the boundary information `rect` provided by the text widget.

The `rect` struct array is structured as follows:

```
typedef struct gui_text_rect
{
    int16_t x1;
    int16_t y1;
    int16_t x2;
    int16_t y2;
    int16_t xboundleft;
    int16_t xboundright;
    int16_t yboundtop;
    int16_t yboundbottom;
} gui_text_rect_t;
```

The `rect` is the display range of the widget passed from the widget layer. In this structure, `x1` and `x2` represent the X-coordinates of the left and right borders, respectively, while `y1` and `y2` represent the Y-coordinates of the top and bottom borders, respectively.

These values are calculated internally by the widget based on its position and size at the time of creation. From the four coordinates of `rect`, you can calculate `rect_w` (width) and `rect_h` (height).

There are also four `bound` values used by the scrolling text widget (`scroll_text`) to handle display boundaries. These `bound` values are currently not used by the regular text widget (`text`).

Developers can add new layout modes as per their requirements.

By enabling the English word wrapping feature (`wordwrap`) via the function `gui_text_wordwrap_set`, the multi-line layout will adhere to English word wrapping rules to prevent words from being split across lines.

Character Rendering

The code for rendering bitmap characters is located in the `rtk_draw_unicode` function in `font_mem.c`.

You can enable matrix operations for the text widget to support text scaling effects; the rendering code for this feature is in `rtk_draw_unicode_matrix` in `font_mem_matrix.c`.

Additionally, you can enable a feature to convert text into an image for achieving complex effects; this rendering code is found in `gui_font_bmp2img_one_char` in `font_mem_img.c`.

The character rendering stage does not involve any layout information; it only reads the glyph information and renders it to the screen buffer.

Each character's rendering is constrained by three boundaries: the widget's boundary, the screen's boundary, and the current character's boundary.

If developers wish to use a special font library for rendering, they need to modify the bitmap data parsing code and draw the pixels into the screen buffer.

4.2.2 API

Defines

FONT_MALLOC_PSRAM(x)

FONT_FREE_PSRAM(x)

FONT_FILE_BMP_FLAG

Functions

uint8_t **gui_font_mem_init**(uint8_t *font_bin_addr)

Initialize the character binary file and store the font and corresponding information in the font list.

Parameters
font_bin_addr – the binary file address of this font type

uint8_t **gui_font_mem_init_ftl**(uint8_t *font_bin_addr)

Initialize the character binary file and store the font and corresponding information in the font list.

Parameters
font_bin_addr – font file address

Returns
uint8_t

uint8_t **gui_font_mem_init_fs**(uint8_t *font_bin_addr)

Initialize the character binary file and store the font and corresponding information in the font list.

Parameters
font_bin_addr – font file address

Returns
uint8_t

uint8_t **gui_font_mem_init_mem**(uint8_t *font_bin_addr)

Initialize the character binary file and store the font and corresponding information in the font list.

Parameters
font_bin_addr – font file address

Returns
uint8_t

uint8_t **gui_font_mem_destroy**(uint8_t *font_bin_addr)

Destroy this flot type in font list.

Parameters
font_bin_addr – font file address

Returns
uint8_t

void **gui_font_mem_load**(*gui_text_t* *text, *gui_text_rect_t* *rect)

Preprocessing of bitmap fonts using internal engines.

Parameters

- **text** – Widget pointer
- **rect** – Widget boundary

void **gui_font_mem_draw**(*gui_text_t* *text, *gui_text_rect_t* *rect)

Drawing of bitmap fonts using internal engine.

Parameters

- **text** – Widget pointer
- **rect** – Widget boundary

void **gui_font_mem_unload**(*gui_text_t* *text)

Post-processing work for drawing bitmap fonts using internal engines.

Parameters
text – Widget pointer

void **gui_font_mem_obj_destroy**(*gui_text_t* *text)

GUI_FONT_SRC_BMP text widget destroy function.

Parameters
text – Widget pointer

uint32_t **gui_get_mem_char_width**(void *content, void *font_bin_addr, *TEXT_CHARSET* charset)

Get the pixel width of the text in the current font file.

Parameters

- **content** – text pointer
- **font_bin_addr** – font file address
- **charset** – text encoding format

Returns
uint32_t

uint32_t **gui_get_mem_utf8_char_width**(void *content, void *font_bin_addr)

Get the pixel width of the utf-8 text in the current font file.

Parameters

- **content** – text pointer
- **font_bin_addr** – font file address

Returns

uint32_t

uint8_t **get_fontlib_by_size**(uint8_t font_size)

Get the fontlib name object.

Parameters

font_size – font size

Returns

uint8_t font lib index

uint8_t **get_fontlib_by_name**(uint8_t *font_file)

Get the fontlib name object.

Parameters

font_file – font file

Returns

uint8_t font lib index

void **gui_font_mem_layout**(*gui_text_t* *text, *gui_text_rect_t* *rect)

text layout by mode

Parameters

- **text** – Widget pointer
- **rect** – Widget boundary

void **gui_font_get_dot_info**(*gui_text_t* *text)

get dot info by utf-8 or utf-16

Parameters

text – Widget pointer

struct **GUI_CHAR_HEAD**

Public Members

uint8_t **char_y**

uint8_t **baseline**

uint8_t **char_w**

uint8_t **char_h**

struct **mem_char_t**

mem char struct start

Public Members

uint32_t **unicode**

int16_t **x**

int16_t **y**

int16_t **w**

int16_t **h**

uint8_t **char_y**

uint8_t **char_w**

uint8_t **char_h**

uint8_t ***dot_addr**

uint8_t ***buf**

gui_img_t ***emoji_img**

struct **MEM_FONT_LIB**

mem char struct end

Public Members

uint8_t ***font_file**

uint8_t **font_size**

FONT_SRC_MODE **type**

uint8_t ***data**

struct **GUI_FONT_HEAD_BMP**

Public Members

uint8_t **head_length**

uint8_t **file_type**

uint8_t **version**[4]

uint8_t **font_size**

uint8_t **rendor_mode**

uint8_t **bold**

uint8_t **italic**

uint8_t **scan_mode**

uint8_t **index_method**

uint8_t **crop**

uint8_t **rsvd**

uint32_t **index_area_size**

uint8_t **font_name_length**

uint8_t ***font_name**

Enums

enum **TEXT_CHARSET**

text rect struct end

text encoding format enum

Values:

enumerator **UTF_8**

enumerator **UTF_16**

enumerator **UTF_16LE**

enumerator **UNICODE_ENCODING**

enumerator **UTF_16BE**

enumerator **UTF_32LE**

enumerator **UTF_32BE**

Functions

uint16_t **process_content_by_charset**(*TEXT_CHARSET* charset_type, uint8_t *content, uint16_t len, uint32_t **p_buf_ptr)

Converts content from a specified charset to Unicode code points.

Parameters

- **charset_type** – The charset type of the content.
- **content** – Input content to be converted.
- **len** – Length of the input content in bytes.
- **p_buf_ptr** – Pointer to the buffer that will hold the Unicode code points.

Returns

The length of the Unicode code points array.

uint32_t **get_len_by_char_num**(uint8_t *utf8, uint32_t char_num)

Get the len by char num object.

Parameters

- **utf8** –
- **char_num** –

Returns

uint32_t

uint32_t **generate_emoji_file_path_from_unicode**(const uint32_t *unicode_buf, uint32_t len, char *file_path)

Function to generate file path based on a given Unicode sequence.

Parameters

- **unicode_buf** –
- **len** –
- **file_path** –

Returns

int

struct **gui_text_rect_t**

text rect struct start

Public Members

int16_t **x1**

int16_t **y1**

int16_t **x2**

int16_t **y2**

int16_t **xboundleft**

int16_t **xboundright**

int16_t **yboundtop**

int16_t **yboundbottom**

4.3 HoneyGUI Porting

HoneyGUI is a lightweight embedded GUI system optimized for Realtek series chips. This document will guide you through compiling the HoneyGUI library on different Realtek chip platforms, including both Armclang and Armcc compiler environments.

4.3.1 Important Notes

- Ensure Keil MDK and CMake are properly installed
- Make sure all required dependencies are installed before compilation
- If compilation errors occur, verify the chip model specification
- **Check compiler path settings:**
  - Armcc compiler default path: C:/Keil_v5/ARM/ARMCC/bin
  - Armclang compiler default path: C:/Keil_v5/ARM/ArmCompilerforEmbedded6.22/bin
  - If installation paths are different, modify compiler paths in CMake configuration accordingly

Fig. 2: Default path for Armcc

Fig. 3: Default path for Armclang

4.3.2 Build Requirements

- CMake 3.15 or above
- Keil MDK 5 or above
- Windows OS

4.3.3 Armcc Compilation

Supported chips:

- RTL8773E (default)
- RTL8763E
- RTL8762G
- RTL8763D

Build steps:

1. Open cmd window in the armcc directory of the project path, generate build files using command `cmake -G "MinGW Makefiles" -DSOC=RTL8763D -B "./temp"`:

```
E:\HoneyGUI\lib\armcc>cmake -G "MinGW Makefiles" -DSOC=RTL8763D -B "./temp"
soc = RTL8763D
-- The C compiler identification is ARMCC 5.6.960
-- The CXX compiler identification is ARMCC 5.6.960
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
...
-- Configuring done (2.7s)
-- Generating done (0.9s)
-- Build files have been written to: E:/HoneyGUI/lib/armcc/temp
```

Note: If chip model is not specified, RTL8773E will be used by default.

2. Enter temp directory to build project, using commands `cd temp` `cmake --build .`:

```
E:\HoneyGUI\lib\armcc>cd temp
E:\HoneyGUI\lib\armcc\temp>cmake --build .
[  1%] Building C object CMakeFiles/gui.dir/E_/HoneyGUI/realgui/3rd/cJSON/cJSON.o
[  2%] Building C object CMakeFiles/gui.dir/E_/HoneyGUI/realgui/3rd/ezXML/ezxml.o
[  3%] Building C object CMakeFiles/gui.dir/E_/HoneyGUI/realgui/3rd/nanovg/base/
↪nanovg.o
...
[100%] Linking C static library gui.lib
[100%] Built target gui
```

3. Install resources, using command `cmake --build . --target install`:

```
E:\HoneyGUI\lib\armcc\temp>cmake --build . --target install
[100%] Built target gui
Install the project...
-- Install configuration: ""
-- Installing: E:/HoneyGUI/lib/armcc/install/lib/gui.lib
...
```

4. **Generated resource file locations:**
    - Header files: E:/HoneyGUI/lib/armcc/install/include
    - Library file: E:/HoneyGUI/lib/armcc/install/lib/gui.lib

4.3.4 Armclang Compilation

Supported chips:

- RTL8762G (default)
- RTL8762D
- RTL8773E
- RTL8773G

Build steps:

1. Open cmd window in the armclang directory of the project path, generate build files using command `cmake -G "MinGW Makefiles" -DSOC=RTL8762G -B "./temp"`:

```
E:\HoneyGUI\lib\armclang>cmake -G "MinGW Makefiles" -DSOC=RTL8762G -B "./temp"
soc = RTL8762G
-- The C compiler identification is ARMClang
-- The CXX compiler identification is ARMClang
...
-- Configuring done
-- Generating done
-- Build files have been written to: E:/HoneyGUI/lib/armclang/temp
```

Note: If chip model is not specified, RTL8762G will be used by default.

2. Enter temp directory to build project, using commands `cd temp` `cmake --build .`:

```
E:\HoneyGUI\lib\armclang>cd temp
E:\HoneyGUI\lib\armclang\temp>cmake --build .
[  0%] Building C object CMakeFiles/gui.dir/...
...
[100%] Built target gui
```

3. Install resources, using command `cmake --build . --target install`:

```
E:\HoneyGUI\lib\armclang\temp>cmake --build . --target install
[100%] Built target gui
Install the project...
-- Installing: E:/HoneyGUI/lib/armclang/install/lib/gui.lib
...
```

4. **Generated resource file locations:**
    - Header files: E:/HoneyGUI/lib/armclang/install/include
    - Library file: E:/HoneyGUI/lib/armclang/install/lib/gui.lib

4.3.5 Project Porting Example

This example demonstrates porting to RTL8773GWP dashboard project.

1. Copy the compiled resource files to the project directory:
    - Copy header files (.h) to the project resource directory
    - Copy library file (gui.lib) to the project resource directory
2. Project Configuration:
    - Add header file path in Keil MDK
    - Link gui.lib in project settings

Fig. 4: Copy header files to project directory

Fig. 5: Link library file to project directory

CHAPTER

FIVE

SAMPLES

We have provided some example applications to help everyone become familiar with using this environment. The sample program will continue to increase.You can choose from the following configurations.The configuration file is `menu_config.h`.

Fig. 1: Configuration Selection

The entry point for any application is:

```
GUI_INIT_APP_EXPORT(app_init);
```

5.1 Calculator

This example demostrates how to develop a simple "Calculator APP", from which you can learn and understand the basic methods and processes of developing a ui application. The "Calculator" works just like a traditional calculator, using button widget for user input and text widget for display. Watch the demo video below to see its full functionality.

5.1.1 Source File

To help learn and be familiar with the development, you can find all source files you may need in path `realguiexamplescreen_448_368`. The source file for this demostration is `app_calculator.c`, you can find it in the path mentioned for more details.

5.1.2 Two Steps

1. Declare the app structure

The app structure saves all the information of ui. Developers should initialize the app structure with the app name and ui design function.

```
#include <gui_app.h>
static void app_calculator_ui_design(gui_app_t *app);

static gui_app_t calculator =
{
    .screen =
    {
```

(continues on next page)

(continued from previous page)

```
        .name = "calculator",
    },
    .ui_design = app_calculator_ui_design,
};

/*
* Public API to get app structure
*/
gui_app_t *get_app_calculator(void)
{
    return &calculator;
}
```

2. Declare the app ui design function

The app ui design function adds all the widgets required to form a complex ui. In this example, we add a window widgets and draw the calculator ui.

```
static void app_calculator_ui_design(gui_app_t *app)
{
    gui_win_t *win = gui_win_create(&app->screen, "back_win", 0, 0, gui_get_screen_
↪width(),
                                    gui_get_screen_height());

    gui_calculator_create(win, "calculator", 0, 0, 454, 454);
}
```

5.2 86Box

This example demostrates how to develop a RealUI 86Box APP, from which you can learn and understand the basic methods and processes of developing a ui application.

5.2.1 Source File

- APP package `realguiexamplescreen_480_480rootappbox`
- UI Design project `RVisualDesigner-v1.0.5.0Demo480x480boxbox480x480.rtkprj`

5.2.2 UI Design

RVisualDesigner

- RealUI 86Box utilizes *RVisualDesigner* to complete UI design. For the first-time usage of *RVisualDesigner*, please refer to `RVisualDesigner-v1.0.5.0RTKIOT Visual Designer User Guide EN.pdf` to obtain a detailed development guide.
- Find and open the example UI design project in the specified path:
- Click on *Export* and then *Simulate* in succession to complete the export and launch the simulation. Once the simulator window is launched, the 86Box APP icon will be displayed. Clicking on it will take you to the corresponding APP.

- When entering the APP, you will see the same UI content as in the *RVisualDesigner* design project in the simulator window. Therefore, this design mode has the feature of "What You See Is What You Get" (WYSIWYG). Developers can drag widgets from the ToolBox to the canvas to create widgets for the current page. After adding image resources to the project, the widgets can be configured and linked to custom images. The hierarchy relationship between widgets will be displayed through the Widget tree.
- This tool requires adding pictures in advance, and then dragging the widgets in the *ToolBox* to the middle screen to lay out the same UI as the current *Widget tree*.

5.2.3 Javascript

- Non-default effects and logic for widgets currently need to be implemented by developers using JavaScript in the current version. For example, control interactions include switch widgets switching images on click, tab widgets sliding , etc. Please refer to *JavaScript syntax* to learn more about the JavaScript-based UI development approach.

Gestures

In the JS file `realguiexamplescreen_480_480rootappboxbox.js`, the control and interaction logic of the UI is implemented.

Light Control Switch

- The callback functions for opening and closing the switch widget named "kitchen_switch" are registered sequentially. When the switch widget "kitchen_switch" is opened, its callback function `led1OnFunc()` will be triggered and called.
- The control of the lights in this example is abstracted as a *Gpio* object. Each light corresponds to a *Gpio* object, and its value is assigned using the `writeSync()` function, which is defined in the underlying layer to accommodate different smart home communication control protocols and control methods.

```
// define an Gpio object
var LED1 = new Gpio(0, 'out');
function led1OnFunc(params) {
        if (sleep_flag) {
                LED1.writeSync(0)
        }
}

// register callback function
sw.getElementById('living_switch')
sw.switch_on(led1OnFunc)
sw.switch_off(led1OffFunc)
```

Tab Jumping Switch

1. Register a tab slide callback for the tabview widget. When the tab is changed by sliding, update the current tab index and synchronize the UI display state.
2. Register a jump control callback function for each switch that controls the navigation. When called back, pass the index value as a parameter to indicate the tab to be navigated to.
3. In the callback function, use the `jump()` function to navigate and synchronize the UI display state.

```
tab.getElementById('tabview0')
var tabJump = {
        cur_tab_x: 0,
        nxt_tab_x: 0,
        cur_tab_y: 0,
        nxt_tab_y: 0
}
function sw_jump_tab(params) {
        // console.log('jump', params)
        tabJump.nxt_tab_x = params

        if(tabJump.nxt_tab_x != tabJump.cur_tab_x)
        {
                sw_jump_turnoff();
                tab.jump(params);
                tabJump.cur_tab_x = tabJump.nxt_tab_x
        }
}

function sw_jump_keep_on(params) {
        // console.log('sw_jump_keep_on ', params)
        Id_prefix = 'sw_tab';
        if(params == tabJump.nxt_tab_x)
        {
                sw.getElementById(sw_getId(params));
                sw.turnOn();
        }
}

function tab_slide(params) {
        // console.log('tab_slide')
        var cur_tab = tab.getCurTab()

        tabJump.nxt_tab_x = cur_tab.x;
        sw_turnOn(tabJump.nxt_tab_x);
        sw_turnOff(tabJump.cur_tab_x);
        tabJump.cur_tab_x = cur_tab.x;
}

// tab change
tab.onChange(tab_slide)

// jump tab0
sw.getElementById('sw_tab0')
sw.onOn(sw_jump_tab, 0)
sw.onOff(sw_jump_keep_on, 0)
```

5.3 LiteGFX

5.3.1 QuDai Introduction

QuDai Technology is a software service company that leverages its self-developed LiteGfx framework to fully harness the performance of various chips, providing customers with cross-platform, one-stop GUI solutions and a plethora of dazzling visual effects products. By utilizing our proprietary 2.5D effects framework, we simulate 3D technology and integrate particle system physics engine technology. All 2.5D effects are embedded within LiteGfx Designer, allowing customers to easily use and personalize them to create unique visual identities. QuDai Technology will continuously enrich its product portfolio in 2.5D technology, helping clients stand out in the fiercely competitive market. We firmly believe that excellent visual design is the key to enhancing a company's brand value and market competitiveness. .. raw:: html

5.3.2 Source Code

QuDai components are integrated into HoneyGUI as a third-party library and used as a Widget in the RealGUI engine. This integration includes three main parts: core library, control adaptation, and platform support.

Source code path: `HoneyGUI\realgui\3rd\litegfx`

```
|-HoneyGUI-Adapt
|      gui_widget_litegfx.c
|      gui_widget_litegfx.h
```

(continues on next page)

(continued from previous page)

```
|       tab_app_energybox.c
|       tab_app_notifications.c
|       tab_app_prsim.c
|       tab_app_prsim_refl.c
|       tab_app_soccer.c
|       tab_watchface_butterfly.c
|       tab_watchface_digitclock.c
|       tab_watchface_flowerfall.c
|       tab_watchface_windmill.c
|
|-platform
|       lx_platform_log.c
|       lx_platform_log.h
|       lx_platform_memory.c
|       lx_platform_memory.h
|       lx_platform_new.cpp
|       lx_platform_time.c
|       lx_platform_time.h
|
|-vglite
    |   liblx_vglite_gcc.a
    |   lx_vglite.lib
    |
    |-include
        |-interface
                lx_vglite_api.h
```

Widget Adaptation Layer

The code in this directory is a wrapper layer provided by QuDai to adapt to HoneyGUI. The files `gui_widget_litegfx.c/h` are for the widgets, while files starting with `tab_` are sample code. Users can refer to these sample codes to implement their own upper-layer effects. Note that essentially, using this method still involves calling the RealGUI engine to display related effects. This widget supports the tiled approach for FrameBuffer.

Platform Adaptation Layer

[illegible]C++[illegible]new[illegible]

Core Lib

It provides the Windows GCC version of `liblx_vglite_gcc.a` and the embedded environment version of `lx_vglite.lib`. Please pay attention to the compiler version.

5.4 Status Bar

- This is a new style status bar. In the non-pull-down state, only the real-time time in small fonts is displayed at the top of the screen.
- Click on the top to pull down the status bar. When pulling down, the mask color gradually becomes opaque and the time text becomes larger.
- After pulling down to a certain extent, the status bar becomes fully expanded, and the date and message notification will be displayed.

5.4.1 Implementation

File

Function `static void status_bar(void *parent, gui_obj_t *ignore_gesture)` is located in file `realgui\example\screen_454_454\gui_menu\apps_in_menu.c`.

Design

- In this status bar, the window widget is the root node. A white semi-transparent background of the status bar is drawn using the rectangle drawing function. Three text boxes are nested, representing time, date, and notification messages, respectively. Among them, the time text box uses a function to cache into an image because the time display needs to be scaled. The rectangular background and the text of the date and notification messages are initially hidden. Touch screen interactive effects are implemented in the animation callback function of the root node window widget.
- In the animation callback function of the window, first update the contents of the text box for time and date to real-time time and date, in the formats of "07:55" and "Tue, Apr 16" respectively. Then, read touchpad data, determining the display effect of the status bar based on current touch screen information such as gestures, for instance, whether to hide the background, whether to hide the date and notifications, change background transparency, time text box reduction scale, and so on.
- The `status_bar` function has a parameter `ignore_gesture`, which takes the pointer to a widget. This parameter is used to resolve conflicts between gestures on the widget and the status bar. When such a conflict is encountered, the gesture interaction of the respective widget is deactivated through this piece of code: `if (ignore_gesture) { ignore_gesture->gesture = 1; }`. Here, setting the `gesture` attribute to '1' turns off the gesture response of the widget in question.

5.5 Fruit Ninja

This example demonstrates how to develop a simple "Fruit Ninja" APP, from which you can learn and understand the basic methods and processes of developing a UI application. Earn points by cutting fruits until you cut a bomb and the game is over. Watch the demo video below to see its full functionality.

5.5.1 Requirements

Refer to the Installation section of *Get Started* .

5.5.2 Source File

To help learn and be familiar with the development, you can find all source files you may need in path `realgui\example\screen_454_454`. The source file for this demostration is `app_fruit_ninja_box2d.cpp`, you can find it in the path mentioned for more details.

5.5.3 Configurations

```
#define SCREEN_WIDTH 454    // Set according to the screen width
#define SCREEN_HEIGHT 454   // Set according to the screen height
#define HEIGHT_OFFSET 100   // Set the screen height offset for refreshing fruit from␣
↪the bottom of the screen
```

5.5.4 Usage Steps

Step 1: Declare the app ui design function

```
/**
* @brief Start Fruit Ninja APP by creating a window,
*        setting the animation function of the window
*        and initializing some variables.
* @param obj The parent widget where the APP's window is nested.
*/
void fruit_ninja_design(gui_obj_t *obj)

void app_fruit_ninja_design(gui_obj_t *obj)
{
   app_fruit_ninja::fruit_ninja_design(obj);
}
```

Step 2: Call function

```
extern void app_fruit_ninja_design(gui_obj_t *obj);
app_fruit_ninja_design(GUI_APP_ROOT_SCREEN);
```

5.5.5 Design Ideas

- In this app, box2d was used to create solids to simulate the movement of objects in a gravitational environment, given parameters such as the initial velocity of the x-axis and y-axis during initialization.

```
/* Add dynamic bodys */
b2BodyDef ballBodyDef;
ballBodyDef.type = b2_dynamicBody;
ballBodyDef.position.Set(4, SCREEN_HEIGHT + HEIGHT_OFFSET * P2M);
ballBodyDef.angularVelocity = -314;    //-PI rad/s
ballBodyDef.linearVelocity.Set(10, -20); // Move up
body_st = world.CreateBody(&ballBodyDef);
```

- The radius of the solid is set to a small value in order to minimize the effect of objects colliding with each other, since mutual collisions are detrimental to the gameplay.

```
/* Creat body shape and attach the shape to the Body */
b2CircleShape circleShape;
circleShape.m_radius = 0.2; // Small radius reducing the impact of␣
↪collisions
```

- The position and rotation angle of the fruits (and bomb) are updated in the callback function using the solid's center point mapping and displayed in the image component. The position and initial velocity of the solid is reset when the position of the fruit is outside the display interface.

```
/* Get the position of the ball then set the image location and rotate␣
↪it on the GUI */
b2Vec2 position = body_st->GetPosition();
if (position_refresh((int)(position.x * M2P - RADIUS_ST), (int)(position.
↪y * M2P - RADIUS_ST),
                     img_strawberry, body_st))
{
   gui_img_set_attribute(img_strawberry, "img_strawberry", FRUIT_NINJA_
↪STRAWBERRY_BIN,
                         img_strawberry->base.x, img_strawberry->base.
↪y);
   fruit_cut_flag[0] = false;
   gui_img_set_location(img_cut_arry[0], 0, SCREEN_HEIGHT + HEIGHT_
↪OFFSET);
}
```

- Cutting fruit uses the structure touch_info, detecting the touch point release indicates the completion of a cut (to get the initial point of the touch screen and the displacement of the x-axis and y-axis), and the content of the cut will be judged.

```
/* Cutting judgment */
GUI_TOUCHPAD_IMPORT_AS_TP // Get touchpoint
if (tp->released)
{
   bool bomb_flag = cutting_judgment(win, img_strawberry, img_banana,␣
```

(continues on next page)

(continued from previous page)

```
↪img_peach, img_watermelon,
                                        img_bomb, tp, img_cut_arry, fruit_
↪cut_flag);
}
```

- If there are two intersection points between the cut line and the picture rectangle, it means that the cut is valid.

```
line_has_two_intersections_with_rectangle(img_coordinate, img_w, img_h,␣
↪tp_start, tp_end,
                                                              img_rotate_
↪angle);
```

- Note that when calculating the intersection point, the rotated endpoint information of the picture needs to be brought into the calculation of the rotation angle to be consistent with the display, so that the accuracy of the cutting judgment can be improved.

```
/* Calculate the rectangle's four rotated points */
Point rotated_rect_min = rotate_point(rect_min, center, angle); // Left-
↪up
Point rotated_rect_max = rotate_point(rect_max, center, angle); // Right-
↪down
Point rotated_rect_p2 = rotate_point(rect_p2, center, angle); // Left-
↪down
Point rotated_rect_p3 = rotate_point(rect_p3, center, angle); // Right-
↪top
```

- Update the fruit picture to two pictures after cutting (corresponding to two gui_img_t pointers) and count the score. Multiple different objects can be cut in a single cut.

```
/* Refresh half-cut fruits position */
if (fruit_cut_flag[0])
{
   gui_img_set_location(img_cut_arry[0],  GUI_BASE(img_strawberry)->x +␣
↪10,
                            GUI_BASE(img_strawberry)->y + 10);
   gui_img_rotation(img_cut_arry[0], gui_img_get_transform_degrees(img_
↪strawberry),
   gui_img_get_width(img_cut_arry[0]) / 2,
   gui_img_get_height(img_cut_arry[0]) / 2);
}
```

- Note that a flag can be used to mark the cut status of the fruit to prevent scoring errors as well as to facilitate updating the position of the cut picture.
- When the cut fruit moves outside the display it will reset the position and initial velocity of the solid and restore the cutting effect.

```
gui_img_set_attribute(img_strawberry, "img_strawberry", FRUIT_NINJA_
↪STRAWBERRY_BIN,
                                  img_strawberry->base.x, img_strawberry->
↪base.y);
fruit_cut_flag[0] = false;
gui_img_set_location(img_cut_arry[0], 0, SCREEN_HEIGHT + HEIGHT_OFFSET);
```

5.6 Music Player

- UI design: Figma - Music Mobile App UI
- Intuitive Three-Layer Design: Navigate effortlessly between three distinct interfaces. The central interface features a sleek display of the current track's album cover along with essential playback widgets.
- Swipe Navigation: With a simple swipe, transition to the top interface to access your song list.
- Lyric Display: Swipe down to reveal the lyrics interface, which is a full-screen display of lyrics, synchronized with the music.
- Smooth Animation: Enjoy a beautiful and fluid transition between interfaces with unique zoom animation, bringing the album cover to life as you switch between viewing options.

5.6.1 Implementation

Code

Function `void app_music_ui_design(gui_obj_t *obj)` is located in file `realgui/example/screen_454_454/gui_menu/app_music.cpp`.

Widgets Tree Design

Fig. 2: Widgets Tree Design

5.7 Timer

- This application features two interfaces Timer & Stopwatch, easily switched with a tap on the two buttons at the bottom.
- Timer Interface: Start the timer with a tap, and watch the seconds increment on the screen.
- Stopwatch Interface: Select your start time using three adjustable rollers for hours, minutes, and seconds. Upon starting, an animation begins as your selected time centers and begins countdown.

5.7.1 Implementation

Code

Function `app_clock_ui_design` is located in file `realgui/example/screen_454_454/gui_menu/app_clock.c`.

Widgets Tree Design

Fig. 3: Timer Widgets Tree Design

5.8 Watchface Market

- Watchface UI design: Figma - Watch Face UI Screens
- This application makes it easy to browse and install new watch faces to suit your style.
- Easy Browsing: Navigate through a clear, two-column layout of watch face previews. Scroll up and down to see your options.
- Preview and Select: Tap any watch face preview to see a fade-out effect, then select it to switch your watch to the new face.
- Quick Access: Long-press your current watch face to quickly open the Face Market App and explore new designs.
- Simple Installation: To install a new watch face, just copy the face package to the specified folder on your smartwatch.

5.8.1 Implementation

Code

Function `GUI_APP_ENTRY(APP_WATCHFACE_MARKET)` is located in file `realgui/example/screen_454_454/gui_menu/watchface_market.c`.

Widgets Tree Design

Fig. 4: Market Tree Design

CHAPTER

SIX

TOOL

In UI design, it is necessary to use Image Convert Tool or Font Convert Tool to convert images or fonts into binary files. Then, Pack Tool is used to package all the UI resource files, and finally, MP Tool is used for burning. This section will introduce the usage of these four tools.

6.1 Image Convert Tool

6.1.1 Image Format Conversion

Convert pictures in various formats into RGB raw pictures

- Open the converter. Please refer to this section for the download link of the image conversion tool: *Tool*.
- The operation steps and detailed instructions are as follows:

1. Open the image folder.
2. Open the settings.
3. Select the output folder.
4. Check the color information header.
5. Choose the image format to be configured.
6. Set the conversion parameters.
7. Convert.

Configuration

- Color head : BeeGUI wants this head to display.
- Big-endian : Whether the input image is big-endian.
- Compress : Enable image compression.
- MixAlphaChannel Flag : Whether to mix alpha channel to rgb when converting rgba to rgb or rgb565.
- Scan Mode : Select whether the scan direction is horizontal or vertical, BeeGUI only wants horizontal.
- Color Space : Select colorSpace (RGB565, RGBA, BINARY…), BeeGUI can display all of them.

Color Space

- RGB565: Colorful but with low rendering cost and storage. 2 bytes per pixel.

Red	Green	Blue
5bit	6bit	5bit

- ARGB8565: 24-bit ARGB mode.

Opacity	Red	Green	Blue
8bit	5bit	6bit	5bit

- RTKRGAB: 16-bit RGAB mode.

Red	Green	Opacity	Blue
5bit	5bit	1bit	5bit

- RGB : 24-bit RGB mode. 3 bytes per pixel.

Red	Green	Blue
8bit	8bit	8bit

- ARGB : True color with opacity. Enhance the display quality with transparency effects. 4 bytes per pixel.

Opacity	Red	Green	Blue
8bit	8bit	8bit	8bit

- BINARY : Use one bit for a pixel.
- RTKARGB8565 : RTK 24-bit ARGB8565 mode.

Opacity	Opacity	...	Red	Green	Blue	Red	Green	Blue	...
8bit	8bit	…	5bit	6bit	5bit	5bit	6bit	5bit	…

6.1.2 Output Files

The following files will be generated.

By using the image conversion tool, we can convert the three `JPG files a, b, and c` into three `binary files a, b, and c`.

Place the `binary file` into the `root` folder of the packaging directory. For the packaging process, please refer to the *Pack Tool* section.

6.2 Font Convert Tool

Font conversion tool features: Obtain the Unicode code corresponding to all characters to be converted from the standard internal code table (codepage file), custom Unicode code table (or supplementary code table .txt file, custom .cst file), and find the vector font data corresponding to characters according to the Unicode code from the font file (such as .ttf). Convert to a bitmap, and the output is a .bin file.

6.2.1 Font Bin Generation

Please refer to the following steps for how to generate files:

1. Copy the Font library file to the directory `\Font Convert Tool\font`.

2. Please refer to the documentation under the directory `\Font Convert Tool\doc` for the specific meanings of each parameter to configure font parameters by editing FontConfig.json.

3. Please open setting.ini and modify the optional configuration items.

4. Double-click fontDirctionary.exe and the font bin will be generated.

6.2.2 FontConfig.json Parameter Description

Table 1: FontConfig.json parameter description

Field name	Field meaning
codePages	A list of selected characters arranged in a specific order for the text of a language. Multiple sets can be configured.
cstPaths	Binary Unicode code point CST file path. Multiple paths can be set.
customerVals	User-defined continuous Unicode characters. Multiple groups can be set.
firstVal	The starting value of a custom continuous Unicode character.
range	A custom number of consecutive Unicode character.
mappingPaths	User-defined Unicode character set file path. Multiple groups can be set.
fontSet	Used to specify font-related settings to be converted.
bold	Specifies whether converted characters are bolded.
italic	Specifies whether the converted characters are slanted.
scanMode	Specifies how the converted character data is saved. If the value is "H", the fonts are saved by row; if the value is "V", the fonts are saved by column.
fontSize	Specifies the converted character size.
font	Specifies the font file to use for the conversion.
renderMode	Specifies how many bits are used to represent a pixel in the converted character bitmap. Supports 1/2/4/8.
indexMethod	Specifies the index mode of the re-index area of the output bin file after conversion, and fills the address index with 0; offset index 1. When the number of characters exceeds 100, it is recommended to choose index mode 0.
crop	Compresses font file size. Always on is recommended. Currently only crop with IndexMethod=0 is supported.

6.2.3 Setting.ini Parameter Description

Table 2: Setting.ini parameter description

gamma	1	The gamma value is a parameter used to describe the nonlinear relationship between input pixel values and output brightness. The higher the value, the higher the text brightness.
rotate	0	Font Rotation Angle. 0: No rotation. 1: Rotate 90° clockwise. 2: Rotate 90° counterclockwise.

6.3 Pack Tool

6.3.1 RTL87x2G and RTL8762D

RTL87x2G is the abbreviation of a series IC type.

The packaging process for the RTL87x2G and RTL8762D is the same. Take RTL8762G as an example as follows.

Before starting, select the appropriate demo under the sdk directory (`\subsys\gui\realgui\example\screen_800_480\root_image_800_480`), or create a new packaging directory based on the example. Then copy the `bat and py scripts` to that directory, ensuring that the `root folder` and the `bat and py scripts` exist under the directory.

1. Copy `resource` all the generated bin files to the `root` folder.
2. Double-click the batch file (`.bat`) to run it, which will execute the packaging process and generate `.bin` file and `.h` file.
3. The `.h` is the address offset of each file in the file system, which can be accessed directly without using the file system. Before developing gui code, please add the folder containing `.h` to the include directory.
4. Use the MPTool tool to burn the `.bin` file into Flash memory.

6.3.2 RTL8763E and RTL8773DO

RTL8763E is the name of a series IC type, including RTL8763EWE-VP/RTL8763EW-VC. The packaging process for RTL877DO is similar to that of RTL8763E.

Before starting, navigate to the SDK directory (`\tool\Gadgets\gui_package_tool`) and choose the appropriate IC directory. Select the `8763E` directory for `RTL8763EW` and the `87x3D` directory for `RTL8773DO`.

The process for generating user data is as follows:

1. Copy all the generated bin files to the folder `\tool\Gadgets\gui_package_tool\8763E\root`.
2. Double-click `gen_root_image.bat` in the `\tool\Gadgets\gui_package_tool\8763E` directory to execute the script and generate an image of the root folder. A new `.bin` file and `.h` file will appear in the directory.
3. Between them, `.bin` is the image file, and `.h` is the address offset of each file in the file system, which can be accessed directly without using the file system.

6.3.3 RTL8773E

RTL8773E is the name of a series IC type, including RTL8773EWE/RTL8773EWE-VP. The user data packaging process is as follows:

Generate Root Bin

1. Copy generated images bin to this folder `\src\app\watch\gui_application\root_image\root\8773e_watch` and Copy generated font bin to this folder `\src\app\watch\gui_application\root_image\root\font`.
2. Modify build address: You need to adjust the address to `0x238b400` by modifying this file `mkromfs_0x4400000.bat` (`python_bin_mkromfs_0x4400000.py --binary --addr 0x238b400 root root(0x4400000).bin`). The `--addr` corresponds to the flash map userdata address +0x400 (image header size)
3. Double-click `mkromfs_0x4400000.bat` in the `\src\app\watch\gui_application\root_image` directory to execute the script and generate an image of the root folder. A new bin `root(0x4400000).bin` file and h file `ui_resource.h` will appear in the directory.
4. Between them, `.bin` is the image file, and `.h` is the address offset of each file in the file system, which can be accessed directly without using the file system.

Note: The generated ui_resource.h requires the following code to be added manually

```
#if defined _WIN32
#else
#include "flash_map.h"

#define   MUSIC_NAME_BIN_ADDR       APP_DEFINED_SECTION_ADDR
#define   MUSIC_HEADER_BIN_ADDR     (MUSIC_NAME_BIN_ADDR + 0xA000)
#define   MUSIC_NAME_BIN_SIZE       (MUSIC_HEADER_BIN_ADDR - MUSIC_NAME_BIN_ADDR)
#define   MUSIC_HEADER_BIN_SIZE     0x5000
#endif
```

Adding Header Information

Using the MPPG Tool to add header information to user data files, the process is as follows:

1. In the menu, select *Tool‣Prepend header for user data*.
2. Add the path to `flash_map.ini`.
3. Add the path to the user data file (`root_xx.bin`).
4. Generate the burnable user data file.

Note: The Max size must be larger than the Actual size; otherwise, the user data size in the flash_map needs to be changed.

6.4 MP Tool

MP Tool supports debugging mode and batch production mode, integrating packaging and flash map generation functions.

- Debug Mode: Offers developers a platform for debugging and feature development.
- MP Mode: Provides an array of capabilities, including the ability to program up to 8 devices concurrently and modify the device's Bluetooth address.

6.4.1 Download to the EVB

Select the chip type and language in the MP Tool startup interface, taking RTL8762G as an example.

Fig. 1: MP Tool Startup Interface

- Load the necessary files for burning, including flash map, System Config File, APP Image, etc.
- Select User Data.

Fig. 2: MP Tool Main Interface

- Download the generated image file to the specified address (such as the file system mount address), where the 8762G address is 0x04400000.

Fig. 3: User Data Loading Interface

- After the file preparation is completed, first check the UART port. If it is normal, it will display *Ready*. Then open the UART port and display *OK*. Once this is done, click *Download* to start the burning process.

Fig. 4: Enter the Burning Mode Interface

CHAPTER

SEVEN

DESIGN SPEC

This section describes the workflow of the RealUI system, which involves the process from input data to displaying on the LCD.

7.1 RealUI System

RealUI system is an efficient embedded solution for display projects based on HoneyGUI.

7.1.1 RealUI Workflow

The workflow of the RealUI system is mainly divided into four steps.

System

System initialization mainly includes the initialization of the system clock, the initialization of peripherals and the initialization of other modules of the project, such as *PSRAM*, *LCD*, *TP*, and Bluetooth.

GUI Server

First, the parts of the GUI port that have been filled in advance are initialized, including the operating system, display, input, and file system. The GUI server thread is then created and the GUI server runs continuously in GUI thread.

GUI Application

A GUI application is a series of display interfaces consisting of multiple widgets. A GUI app is a series of display interfaces consisting of multiple widgets. In order to run a GUI APP, it needed to be started.

GUI Server Task

GUI server is the running function of GUI task, and its specific running process is divided into six parts:

1. GUI *APP* EXIST: First, the currently running GUI APP needs to be obtained. When the GUI detects that there is a running GUI APP, it will proceed to the next step;
2. GET *LCD* DATA: Get real-time information about the screen;
3. GET *TP* DATA: Get real-time information about the touchpad and run the touch algorithm;
4. GET *KB* DATA: Get real-time information about the keyboard and run the keyboard algorithm;
5. OBJ DRAW: Drawing widgets in the APP, including functional operations and image processing.
6. UPDATE *FB*: Delivers the drawn results to the screen.

More detailed operation of the GUI APP can be found in the online documentation.

7.2 Input Subsystem

The UI system can receive input from other peripherals in the device, typical input devices are touchpads and buttons.

This chapter describes how to use input devices in the UI system and describes the processing of input information in detail.

7.2.1 Touchpad

The touchpad is one of the most commonly used input devices, and most of the time, the touchpad is integrated into the display panel. The workflow for touch information is shown in the figure below.

Fig. 1: Touchpad Information Flow

Touchpad Hardware and Driver

Although different touchpad chips have different message data structures, the message always contains the touch status and the coordinates of the touch point. In order to transmit coordinate information, a data bus is needed, and *I2C* is the most commonly used data bus between touch chips and microprocessors.

In addition, different touch chips need to use different drivers according to their specifications, which needs to be ported.

Get Touchpad Data

In the `port_touchpad_get_data` function, the touch information will be fetched in `drv_touch_read`, processed briefly, and fed into the touch algorithm handler as raw data.

```
struct gui_touch_port_data *port_touchpad_get_data()
{
    uint16_t x = 0;
    uint16_t y = 0;
    bool pressing = 0;
    if (drv_touch_read(&x, &y, &pressing) == false)
    {
```

(continues on next page)

(continued from previous page)

```
        return NULL;
    }
    if (pressing == true)
    {
        raw_data.event = 2;
    }
    else
    {
        raw_data.event = 1;
    }
    raw_data.timestamp_ms = os_sys_tick_get();
    raw_data.width = 0;
    raw_data.x_coordinate = x;
    raw_data.y_coordinate = y;
    //gui_log("event = %d, x = %d, y = %d, \n", raw_data.event, raw_data.x_coordinate,
↪ raw_data.y_coordinate);
    return &raw_data;
}
```

The data structure of the raw data is `gui_touch_port_data_t`.

Touchpad Algorithm Processor

The code implementation of the touchpad algorithm processing is in the `tp_algo_process` function. Gesture recognition is performed by judging changes in X-axis and Y-axis coordinate data, as well as touch time. The input types obtained after the algorithm processing are as follows.

```
typedef enum
{
    TOUCH_INIT                  = 0x100,
    TOUCH_HOLD_X,
    TOUCH_HOLD_Y,
    TOUCH_SHORT,
    TOUCH_LONG,
    TOUCH_DOUBLE,
    TOUCH_ORIGIN_FROM_X,
    TOUCH_ORIGIN_FROM_Y,
    TOUCH_LEFT_SLIDE,
    TOUCH_RIGHT_SLIDE,
    TOUCH_UP_SLIDE,
    TOUCH_DOWN_SLIDE,
    TOUCH_SHORT_BUTTON,
    TOUCH_LONG_BUTTON,
    TOUCH_UP_SLIDE_TWO_PAGE,
    TOUCH_DOWN_SLIDE_TWO_PAGE,
    TOUCH_INVALIDE              = 0x1FF,

    KB_INIT                     = 0x200,
    KB_SHORT                    = 0x201,
    KB_LONG                     = 0x202,
    KB_INVALIDE                 = 0x2FF,

    WHEEL_INIT                  = 0x300,
    WHEEL_ING,
    WHEEL_FINISHED,
```

(continues on next page)

(continued from previous page)

```
    WHEEL_INVALIDE              = 0x3FF,
} T_GUI_INPUT_TYPE;
```

The algorithm processor fills in the `touch_info_t` structure, which is available to all widgets.

Widget Response

Some widgets can respond to touchpad information, such as window widgets, button widgets, tab widgets, curtain widgets and progress bar widgets. Among them, windows and buttons mainly respond to click events, tab, curtain and progress bar mainly respond to swipe events. In addition, the display of tabs, curtains, and progress bars also depends on the touch real-time coordinates in the `touch_info_t` structure.

Most of the widgets that process touch information are located in the corresponding preparation function, such as `win_prepare`. Use `tp_get_info` to get touch information.

At the application level, different callback functions can be bound to different kinds of events in the following ways.

```
gui_img_t *hour;
gui_img_t *minute;
gui_img_t *second;
void show_clock(void *obj, gui_event_t e)
{
    if (GET_BASE(hour) == false)
    {
        gui_obj_show(hour, false);
        gui_obj_show(minute, false);
        gui_obj_show(second, false);
        gui_img_set_attribute((gui_img_t *)home_bg, "home_bg", home[1], 0, 0);
    }
    else
    {
        gui_obj_show(hour, true);
        gui_obj_show(minute, true);
        gui_obj_show(second, true);
        gui_img_set_attribute((gui_img_t *)home_bg, "home_bg", home[0], 0, 0);
    }
}
void enter_homelist(void *obj, gui_event_t e)
{
    gui_log("enter_tablist \n");
    gui_app_switch(gui_current_app(), get_app_homelist());
}
void design_tab_home(void *parent)
{
    hour = gui_img_create_from_mem(parent, "hour", TIME_HOUR_BIN, 160, 192, 0, 0);
    minute = gui_img_create_from_mem(parent, "minute", TIME_MUNITE_BIN, 160, 192, 0,␣
↪0);
    second = gui_img_create_from_mem(parent, "second", TIME_SECOND_BIN, 160, 192, 0,␣
↪0);
    gui_win_t *clock = gui_win_create(parent, "clock", 0, 84, 320, 300);
    gui_obj_add_event_cb(clock, (gui_event_cb_t)show_clock, GUI_EVENT_TOUCH_CLICKED,␣
↪NULL);
    gui_obj_add_event_cb(clock, (gui_event_cb_t)enter_homelist, GUI_EVENT_TOUCH_LONG,␣
↪NULL);
}
```

In this example, a window named clock is created first, and when clicked, it executes the `show_clock` function. When prolonged, it executes the `enter_homelist` function.

7.2.2 Keyboard

The workflow for keyboard information is shown in the figure below.

Fig. 2: Keyboard Information Flow

Hardware and Driver

The hardware design and driver program of the keyboard are relatively simple. Here, we will demonstrate this using a single *GPIO*. For information on how to use GPIO, please refer to the instructions in the SDK. You can use the general *API* in `rtl87x2g_gpio.c` or the encapsulated API in `drv_gpio.c` to accomplish the same tasks.

Get Keyboard Data

In the `port_kb_get_data` function, the touch information will be fetched. Users need to fill `port_kb_get_data` according to their functional requirements and fill the structure with keyboard input information.

Keyboard Algorithm Processor

The code implementation of the keyboard algorithm processing is in the `kb_algo_process` function. It can be determined whether the type of input is short press or long press by pressing for a long time. The algorithm processor fills in the `kb_info_t` structure, which is available to all widgets.

Response

There are two ways to respond to the keyboard, one is to respond to the processed key information in the widget such as window, and the other is to respond directly to the press action when the key is received.

The first way is as follow.

```
static void win_prepare(gui_obj_t *obj)
{
    gui_dispdev_t *dc = gui_get_dc();
    touch_info_t *tp = tp_get_info();
    kb_info_t *kb = kb_get_info();
    if (kb->pressed == true)
    {
        gui_obj_enable_event(obj, GUI_EVENT_KB_DOWN_PRESSED);
    }
......
}
```

For the second type, please refer to the GPIO user manual.

7.3 Display Subsystem

The workflow of the display system is very complex, and there are different processes for different UI frameworks and different widgets.

7.3.1 Display Workflow

As the most commonly used UI input source, the image is used here as an example to illustrate the complete workflow from the raw image to the screen, as shown in the figure below.

Due to the different hardware configurations of various types of *IC*, the RTL8772G chip platform is chosen here, and RealUI is used as the UI system to explain the image display workflow.

Fig. 3: Image Display Work Flow

Flash File System

The original image is converted into a file in a special format and then downloaded into flash. Flash is configured with a pseudo-file system that provides image index information to the widget layer.After the simple migration of the file system is complete, the standard file system can be used.

Please read the *Image Convert Tool* section for more information about image conversion.

UI Widget

The image widget is the most basic UI widget used to display images. There are many widgets in the UI system that draw special images based on image widgets.

Here, the image widget loads the image data and reads the image information. It combines the UI design and the behavior of the widget layer to provide image rendering requirements for the acceleration layer. Such as image movement, image reduction and enlargement, image rotation and so on.

In addition, some of the hardware supports powerful *GPU* that can draw widgets with complex transformation effects, such as cube widget, color wheel widget, and so on.

Acceleration Layer

The function of the acceleration layer is to accelerate the UI image drawing process, which is divided into hardware acceleration and software acceleration. In general, hardware acceleration is significantly better than software acceleration, but which one to use depends on the hardware environment in which the UI system is deployed. In addition, different hardware accelerators, also known as graphics processing units (GPU), also have different capabilities. The accelerator receives the drawing task assigned by the UI widget and transfers the completed image to the display buffer.

Buffer

In most embedded systems where *RAM* is limited, RealUI uses a chunked rendering mechanism that requires a display buffer. The display buffer stores the image drawing results of the accelerator and the drawing results of other non-accelerating widgets, and the data are transferred to the frame buffer through *DMA*.

Single frame draw mode can be used when the available RAM can accommodate a full frame, in which case a full frame buffer is used instead of a display buffer.

After configuring the display controller, it will transfer the frame buffer data to the screen, at this time, the screen will display the UI interface.

7.4 Software Accelerate

7.4.1 Overall Flow Chart

The flowchart depicts the image resource processing flow accelerated by software. When processing images, different processing methods are selected based on the compression status and type of image:

- **Cover**: Write the source image data directly to the corresponding position in the frame buffer. Do not perform any processing, just overwrite it.
- **Bypass**: Write the source image data directly to the corresponding position in the frame buffer. Bypass mode is incapable of handling the transparency of images. It applies a global opacity value to the entire image, thereby affecting the overall transparency. When it comes to creating transparency effects, bypass mode is more space-efficient compared to source_over mode.
- **Filter black**: The filtering technique effectively sifts out pixel data with a value of zero from the originating image data, which essentially means that black pixels are precluded from being inscribed into the frame buffer. This mechanism induces much swifter refresh dynamics. Pixels of any color other than black undergo the standard processing method and are duly recorded into the frame buffer.
- **Source_over**: A blending method that combines image color data and frame buffer pixel color data to calculate the final color based on the opacity_value value Sa, and writes it to the corresponding location in the frame buffer. The formula is ((255 - Sa) * D + Sa * S) / 255), where Sa is the opacity_value of the original image, D is the frame buffer pixel data, and S is the source image pixel data.

Fig. 4: Software acceleration

- The img_type can be obtained from the head of the image, where the structure of the image head is as follows.

```
typedef struct gui_rgb_data_head
{
    unsigned char scan : 1;
    unsigned char align : 1;
    unsigned char resize: 2; //0-no resize;1-50%(x&y);2-70%;3-80%
    unsigned char compress: 1;
    unsigned char rsvd : 3;
    char type;
    short w;
    short h;
    char version;
    char rsvd2;
} gui_rgb_data_head_t;
```

- The value of img_type is depicted in the enum below. If the value is IMDC_COMPRESS, it indicates that the image is compressed and enters the rle processing flow; otherwise, it enters the no rle processing flow.

```
typedef enum
{
    RGB565      = 0, //bit[4:0] for Blue, bit[10:5] for Green, bit[15:11]    for Red
    ARGB8565    = 1, //bit[4:0] for Blue, bit[10:5] for Green, bit[15:11]    for Red,␣
↪bit[23:16] for Alpha
    RGB888      = 3, //bit[7:0] for Blue, bit[15:8] for Green, bit[23:16]    for Red
    ARGB8888    = 4, //bit[7:0] for Blue, bit[15:8] for Green, bit[23:16]    for Red,␣
↪bit[21:24] for Alpha
    BINARY      = 5,
    ALPHAMASK   = 9,
    BMP         = 11,
    JPEG        = 12,
    PNG         = 13,
    GIF         = 14,
    RTKARGB8565 = 15,
} GUI_FormatType;
```

- Execute the corresponding blit process based on different blend_mode.

```
typedef enum
{
    IMG_BYPASS_MODE = 0,
    IMG_FILTER_BLACK,
    IMG_SRC_OVER_MODE, //S * Sa + (1 - Sa) * D
    IMG_COVER_MODE,
    IMG_RECT,
} BLEND_MODE_TYPE;
```

- When the image is compressed, it is necessary to obtain the compression header from the address of the compressed data. The algorithm_type parameter of this header contains the actual image type. The types of compressed images are described in the imdc_src_type struct, which includes three types: IMDC_SRC_RGB565, IMDC_SRC_RGB888, and IMDC_SRC_ARGB8888.

```
typedef struct imdc_file_header
{
    struct
    {
        uint8_t algorithm: 2;
        uint8_t feature_1: 2;
        uint8_t feature_2: 2;
        uint8_t pixel_bytes: 2;
    } algorithm_type;
    uint8_t reserved[3];
    uint32_t raw_pic_width;
    uint32_t raw_pic_height;
} imdc_file_header_t;
```

```
typedef enum
{
    IMDC_SRC_RGB565 = 0x04, // 4,
    IMDC_SRC_RGB888 = 0x44, // 68,
    IMDC_SRC_ARGB8888 = 0x84, // 132,

} imdc_src_type;
```

7.4.2 Overview No RLE Cover Mode

The following flow describes the cover mode process for No RLE compressed image. Select a processing method based on the image matrix and the pixel byte of the display device, and write it to the frame buffer.

Fig. 5: Cover Mode Path

- If the matrix is an identity matrix, a blit process without matrix operations is performed; otherwise, a blit process with matrix operations is carried out.
- The dc_bytes_per_pixel is pixel bytes of display device, calculated as dc->bit_depth >> 3, where bit_depth is the bit depth of the display device. Taking a display device with a bit depth of 24 as an example, its pixel bytes are 3.

No RLE Cover

The following flowchart describes the process of writing uncompressed images to a frame buffer in cover mode. Taking the target device image type as RGB565 as an example.

Fig. 6: Cover_blit_2_rgb565

No RLE Cover Matrix

The following flowchart describes the process of writing uncompressed images to a frame buffer using cover mode with matrix operations. Taking the target device image type as RGB565 as an example.

Fig. 7: Cover_matrix_blit_2_rgb565

7.4.3 Overview No RLE Bypass Mode

The following flow describes the bypass mode process for No RLE compressed image. Select a processing method based on the image matrix and the pixel byte of the display device, and write it to the frame buffer.

Fig. 8: Bypass_mode_path

- If the matrix is an identity matrix, a blit process without matrix operations is performed; otherwise, a blit process with matrix operations is carried out.
- The dc_bytes_per_pixel is pixel bytes of display device, calculated as dc->bit_depth >> 3, where bit_depth is the bit depth of the display device. Taking a display device with a bit depth of 24 as an example, its pixel bytes are 3.

No RLE Bypass Mode

The following flowchart describes the process of writing uncompressed images to a frame buffer in bypass mode . Taking the target device image type as RGB565 as an example.

Fig. 9: Bypass_blit_2_rgb565

1. Perform different processing steps based on the img_type.
2. Based on the opacity_value , execute the corresponding operation to write image pixels into the framebuffer.

- If the opacity_value is 0 , the image is not displayed and the process is break.
- If the opacity_value is 255 , convert the source image pixels to RGB565 format and write them to the frame buffer.
- If the opacity_value is between 0 and 255 , perform an alpha blending operation to blend the source image pixels with the corresponding frame buffer pixels. The blending formula is ((255 - Sa) * D + Sa * S) / 255) . Write the blended result to the frame buffer.

No RLE Bypass Matrix

The following flowchart describes the process of writing uncompressed images to a frame buffer using blend mode with matrix operations. Taking the target device image type as RGB565 as an example.

Fig. 10: Bypass_matrix_blit_2_rgb565

1. Perform different processing steps based on the img_type.
2. Perform matrix calculation to map the target area write-in points to image pixels, and obtain the pixel value of the image pixels.
3. Based on the opacity_value, execute the corresponding operation to write image pixels into the framebuffer.

- If the opacity_value is 0, the image is not displayed and the process is break.
- If the opacity_value is 255, convert the source image pixels to RGB565 format and write them to the frame buffer.
- If the opacity_value is between 0 and 255, perform an alpha blending operation to blend the source image pixels with the corresponding frame buffer pixels. The blending formula is ((255 - Sa) * D + Sa * S) / 255). Write the blended result to the frame buffer.

7.4.4 Overview No RLE Filter

The following flow describes the filter mode process for No RLE compressed image. Select a processing method based on the image matrix and the pixel byte of the display device, and write it to the frame buffer.

Fig. 11: Filter_mode_path

No RLE Filter

The following flowchart describes the process of writing uncompressed images to a frame buffer using filter mode. Taking the target device image type as RGB565 as an example.

Fig. 12: Filter_blit_2_rgb565

1. Perform different processing steps based on the img_type.
2. If the pixel value is 0, skip the processing; otherwise, perform the subsequent writing operation.
3. Based on the opacity_value , execute the corresponding operation to write image pixels into the framebuffer.

- If the opacity_value is 0, the image is not displayed and the process is break.
- If the opacity_value is 255, convert the source image pixels to RGB565 format and write them to the frame buffer.
- If the opacity_value is between 0 and 255, perform an alpha blending operation to blend the source image pixels with the corresponding frame buffer pixels. The blending formula is ((255 - Sa) * D + Sa * S) / 255). Write the blended result to the frame buffer.

No RLE Filter Matrix

The following flowchart describes the process of writing uncompressed images to a frame buffer using filter mode with matrix operations . Taking the target device image type as RGB565 as an example.

Fig. 13: Filter_matrix_blit_2_rgb565

1. Perform different processing steps based on the img_type.
2. Perform matrix calculation to map the target area write-in points to image pixels, and obtain the pixel value of the image pixels.
3. If the pixel value is 0, skip the processing; otherwise, perform the subsequent writing operation.
4. Based on the opacity_value, execute the corresponding operation to write image pixels into the framebuffer.

- If the opacity_value is 0, the image is not displayed and the process is break.
- If the opacity_value is 255, convert the source image pixels to RGB565 format and write them to the frame buffer.
- If the opacity_value is between 0 and 255, perform an alpha blending operation to blend the source image pixels with the corresponding frame buffer pixels. The blending formula is ((255 - Sa) * D + Sa * S) / 255). Write the blended result to the frame buffer.

7.4.5 Overview No RLE Source_over

The following flow describes the source_over mode process for No RLE compressed image. Select a processing method based on the image matrix and the pixel byte of the display device, and write it to the frame buffer.

Fig. 14: Alpha_mode_path

No RLE Alpha No Matrix

The following flowchart describes the process of writing uncompressed images to a frame buffer using source_over mode. Taking the target device image type as RGB565 and the source image type as RGB565 as an example.

Fig. 15: Alpha_blit_2_rgb565

Based on the opacity_value , execute the corresponding operation to write image pixels into the framebuffer. - If the opacity_value is 0, the image is not displayed and the process is break. - If the opacity_value is 255, convert the source image pixels to RGB565 format and write them to the frame buffer. - If the opacity_value is between 0 and 255, perform do_blending_acc_2_rgb565_opacity to blend the source image pixels with the corresponding frame buffer pixels. Write the blended result to the frame buffer.

No RLE Alpha Matrix

The following flowchart describes the process of writing uncompressed images to a frame buffer using source_over mode with matrix operations. Taking the target device image type as RGB565 and the source image type as RGB565 as an example.

Fig. 16: Alpha_matrix_blit_2_rgb565

1. Perform matrix calculation to map the target area write-in points to image pixels, and obtain the pixel value of the image pixels.
2. Based on the opacity_value, execute the corresponding operation to write image pixels into the framebuffer.

- If the opacity_value is 0, the image is not displayed and the process is break.
- If the opacity_value is 255, convert the source image pixels to RGB565 format and write them to the frame buffer.
- If the opacity_value is between 0 and 255, perform do_blending_acc_2_rgb565_opacity to blend the source image pixels with the corresponding frame buffer pixels. Write the blended result to the frame buffer.

7.4.6 Overview RLE Cover Mode

The following flow describes the cover mode process for RLE compressed image. Select a processing method based on the image matrix and the pixel byte of the display device, and write it to the frame buffer.

Fig. 17: Rle_cover_mode_path

RLE Cover No Matrix

The following flowchart describes the process of writing compressed images to a frame buffer in cover mode. Taking the target device image type as RGB565 as an example.

Fig. 18: Rle_cover_blit_2_rgb565

1. Perform different processing steps based on the img_type from the head of compression data.
2. Decompress the compressed image data.
3. Write the pixel result to the frame buffer.

RLE Cover Matrix

The following flowchart describes the process of writing compressed images to a frame buffer in cover mode with matrix operations . Taking the target device image type as RGB565 as an example.

Fig. 19: Rle_cover_matrix_blit_2_rgb565

1. Perform different processing steps based on the img_type from the head of compression data.
2. Decompress the compressed image data.
3. Perform matrix calculation to map the target area write-in points to image pixels, and obtain the pixel value of the image pixels.
4. Write the pixel result to the frame buffer.

7.4.7 Overview RLE Bypass Mode

The following flow describes the bypass mode process for RLE compressed image. Select a processing method based on the image matrix and the pixel byte of the display device, and write it to the frame buffer.

Fig. 20: Rle_bypass_mode_path

RLE Bypass No Matrix

The following flowchart describes the process of writing compressed images to a frame buffer in bypass mode . Taking the target device image type as RGB565 as an example.

Fig. 21: Rle_bypass_blit_2_rgb565

1. Perform different processing steps based on the img_type from the head of compression data.
2. Decompress the compressed image data.
3. Based on the opacity_value, execute the corresponding operation to write image pixels into the framebuffer.

- If the opacity_value is 0, the image is not displayed and the process is break.
- If the opacity_value is 255, convert the source image pixels to RGB565 format and write them to the frame buffer.
- If the opacity_value is between 0 and 255, perform an alpha blending operation to blend the source image pixels with the corresponding frame buffer pixels. The blending formula is ((255 - Sa) * D + Sa * S) / 255). Write the blended result to the frame buffer.

RLE Bypass Matrix

The following flowchart describes the process of writing compressed images to a frame buffer in bypass mode with matrix operations. Taking the target device image type as RGB565 as an example.

Fig. 22: Rle_bypass_matrix_blit_2_rgb565

1. Perform different processing steps based on the img_type from the head of compression data.

2. Decompress the compressed image data.
3. Perform matrix calculation to map the target area write-in points to image pixels, and obtain the pixel value of the image pixels.
4. Based on the opacity_value, execute the corresponding operation to write image pixels into the framebuffer.

- If the opacity_value is 0, the image is not displayed and the process is break.
- If the opacity_value is 255, convert the source image pixels to RGB565 format and write them to the frame buffer.
- If the opacity_value is between 0 and 255, perform an alpha blending operation to blend the source image pixels with the corresponding frame buffer pixels. The blending formula is ((255 - Sa) * D + Sa * S) / 255). Write the blended result to the frame buffer.

7.4.8 Overview RLE Filter

The following flow describes the filter mode process for RLE compressed image. Select a processing method based on the image matrix and the pixel byte of the display device, and write it to the frame buffer.

Fig. 23: Rle_filter_mode_path

RLE Filter

The following flowchart describes the process of writing compressed images to a frame buffer in filter mode. Taking the target device image type as RGB565 as an example.

Fig. 24: Rle_filter_blit_2_rgb565

1. Perform different processing steps based on the img_type from the head of compression data.
2. Decompress the compressed image data.
3. If the pixel value is 0, skip the processing; otherwise, perform the subsequent writing operation.
4. Based on the opacity_value, execute the corresponding operation to write image pixels into the framebuffer.

- If the opacity_value is 0, the image is not displayed and the process is break.
- If the opacity_value is 255, convert the source image pixels to RGB565 format and write them to the frame buffer.
- If the opacity_value is between 0 and 255, perform an alpha blending operation to blend the source image pixels with the corresponding frame buffer pixels. The blending formula is ((255 - Sa) * D + Sa * S) / 255). Write the blended result to the frame buffer.

RLE Filter Matrix

The following flowchart describes the process of writing compressed images to a frame buffer in filter mode with matrix operations. Taking the target device image type as RGB565 as an example.

Fig. 25: Rle_filter_matrix_blit_2_rgb565

1. Perform different processing steps based on the img_type from the head of compression data.
2. Decompress the compressed image data.

3. Perform matrix calculation to map the target area write-in points to image pixels, and obtain the pixel value of the image pixels.
4. If the pixel value is 0, skip the processing; otherwise, perform the subsequent writing operation.
5. Based on the opacity_value , execute the corresponding operation to write image pixels into the framebuffer.

- If the opacity_value is 0, the image is not displayed and the process is break.
- If the opacity_value is 255, convert the source image pixels to RGB565 format and write them to the frame buffer.
- If the opacity_value is between 0 and 255, perform an alpha blending operation to blend the source image pixels with the corresponding frame buffer pixels. The blending formula is ((255 - Sa) * D + Sa * S) / 255). Write the blended result to the frame buffer.

7.4.9 Overview RLE Source_over

The following flow describes the source_over mode process for RLE compressed image. Select a processing method based on the image matrix and the pixel byte of the display device, and write it to the frame buffer.

Fig. 26: Rle_alpha_blit_2_rgb565

RLE Source_over No Matrix

The following flowchart describes the process of writing compressed images to a frame buffer in source_over mode. Taking the target device image type as RGB565 as an example.

1. Perform different processing steps based on the img_type from the head of compression data.
2. Decompress the compressed image data.
3. Based on the opacity_value , execute the corresponding operation to write image pixels into the framebuffer.

- If the opacity_value is 0, the image is not displayed and the process is break.
- If the opacity_value is 255: When the source image is in RGB565 format, directly write it to the frame buffer. Otherwise, perform the corresponding do blend operation and write the blend result to the frame buffer.
- If the opacity_value is between 0 and 255, perform the appropriate do_blending operation to blend the source image pixels with the corresponding frame buffer pixels. Write the blended result to the frame buffer.

RLE Source_over Matrix

The following flowchart describes the process of writing compressed images to a frame buffer in source_over mode with matrix operations . Taking the target device image type as RGB565 as an example.

Fig. 27: Rle_alpha_matrix_blit_2_rgb565

1. Perform different processing steps based on the img_type from the head of compression data.
2. Decompress the compressed image data.
3. Perform matrix calculation to map the target area write-in points to image pixels, and obtain the pixel value of the image pixels.

4. Based on the opacity_value , execute the corresponding operation to write image pixels into the framebuffer.

- If the opacity_value is 0, the image is not displayed and the process is break.
- If the opacity value level is 255: When the source image is in RGB565 format, directly write it to the frame buffer. Otherwise, perform the corresponding do blend operation and write the blend result to the frame buffer.
- If the opacity_value is between 0 and 255, perform the appropriate do_blending operation to blend the source image pixels with the corresponding frame buffer pixels. Write the blended result to the frame buffer.

Note: In compressed source_over matrix mode output rle_rgb888 and rle_rgba8888 equivalent to output as rle_rgb565.

7.4.10 Support Input Type and Output Type

Input type	Output type
RGB565	RGB565
RGB888	RGB888
ARGB8888	RLE_ARGB8888
ARGB8565	ARGB8565
RLE_RGB565	RLE_RGB565
RLE_RGB888	RLE_RGB888
RLE_ARGB8888	RLE_ARGB8888
RLE_ARGB8565	RLE_ARGB8565

CHAPTER

EIGHT

FAQ

Some common problems that arise during the use of GUI can be referred to in this chapter.

8.1 Development Environment

8.1.1 Simulator in VSCode

If you encounter problems the first time you run the simulator in VSCode, please check the following configurations in your development environment:

Installing the Appropriate Version of the Toolchain

Simulator in VSCode using MinGW toolchain (refer to *Install compiler* in Get Started), and MinGW version 8.1.0 is recommended, which can be accessed from MinGW v8.1.0 Download. There is no guarantee that all later versions of the MinGW will function properly.

> **Warning:** VSCode currently does not support gdb version higher than v9.2.0 in MinGW. (gdb v8.1 is used in MinGW v8.1.0, which is recommended.)

Adding Toolchain to System Environment Variables

Make sure `C:/mingw64/bin` is already added to system environment variable `Path`.

8.2 Porting

8.2.1 User Data

User Data bin image generation need to consider user data address in `flash_map.h`. Normally the address in generate script is consistent with user data address in `flash_map.h`, and if user data bin need to add image data header due to `mppgtool` requirement, the generate script address must increase by image data header size.

8.2.2 JS Malloc Heap

JS (javascript) is included in GUI module, the heap space JS used may fail to malloc due to resource limitation, so this heap space could relocate on psram if SoC supports psram feature. The specific information can be found in the API `void *context_alloc(size_t size, void *cb_data_p)`.

8.2.3 Feed Watch Dog

GUI task does not support feeding the watch dog, so the app should do this in a hook function registered by the APP and used by GUI. The registered function is `void gui_task_ext_execution_sethook(void (*hook)(void))`.

8.2.4 Not Support FPU

If SoC does not support FPU, some headers and code should be excluded by macros, for example, RTL8763EP.

8.2.5 File System

SoC needs to read image and font resources from flash by file system which should set a start address that is consistent with the address in User Data generation script. GUI has supplied the related file which is `romfs.c` where the start address is `ROMFS_ADDR`.

8.2.6 Flash Setting

Flash speed mode should be set to 4 bit mode; flash clock should be set to a higher value based on chip capabilities.

8.2.7 CPU Frequence

CPU frequency should be set to a higher value based on chip capabilities.

8.2.8 SCONS Version

A specific scons version is required, please use the `pip install scons==4.4.0` command to download.

8.3 Specification

8.3.1 Graphics

Platform	RTL8762D	RTL8772F	RTL87X2G	RTL8763E	PC
RGB565	Y	Y	Y	Y	Y
RGB888	N	Y	Y	N	Y
ARGB8888	N	Y	Y	N	Y
SVG	N	Y	N	N	Y
TTF	N	Y	N	N	Y
DOT font	Y	Y	Y	Y	Y
Vector Graphics	N	Y	N	N	Y
Linear gradient	N	Y	N	N	Y
Radial gradient	N	N	N	N	Y

8.3.2 Memory Usage

RTL8772F Demo

The memory consumption statistics of this demo are as follows.

Module	Cost
Widget	14.56KB
Framebuffer	screenWidth * pixelBytes * Lines
Thread stack	10KB

Widget Memory Usage

Widget	Memory(Byte) on ARM SoC	Memory(Byte) on Win_32 SIM
obj	52	88
img	112	178
win	72	112
page	124	184
tab	88	136
tabview	100	160
button	88	160
text	100	176
scroll_text	120	200
app	92	152
canvas_arc	156	264
canvas_rect	64	104
canvas	60	104
card	72	112
cardview	124	176
colorwheel	72	112

continues on next page

Table 1 – continued from previous page

Widget	Memory(Byte) on ARM SoC	Memory(Byte) on Win_32 SIM
cube	748	928
curtain	60	96
curtainview	120	168
gallery	112	184
grid	100	144
img_live	84	144
img_scope	124	192
stb_img	76	144
kb	108	192
map	196	272
menu_cellular	76	120
multi_level	60	104
pagelist	96	160
pagelistview	64	112
perspective	736	920
progressbar	80	136
qbcode	84	136
scroll_wheel	388	696
seekbar	128	216
simple_img	68	120
svg	96	144
turn_table	128	192
watch_gradient_spot	60	96
wave	72	112
wheel_list	64	112

8.4 How To Increase FPS

8.4.1 Pixel format

Using RGBA/RGB images can get great display effects, but if the FPS is low, then you can use RGB565 format image resources, sacrifice a little effect to get a FPS boost.

8.4.2 Hardware Acceleration

Use hardware acceleration to render images instead of software acceleration whenever possible. Different chip models may have different GPU, please refer to the guidance document in the SDK for details.

8.4.3 Data Transmission Speed

HoneyGUI supports image compression, and some chips have built-in hardware decompression modules. Using hardware decompression modules is very fast, but software decompression requires a certain amount of time. Compressed images can reduce the size of the original image resources, allowing more resources to be stored in user data, and will also reduce the time needed to read from flash.

8.4.4 UI Design

Reducing complexity in the UI, as well as the number and size of images in a single interface, can increase the frame rate. Make sure that every pixel of the image data that needs to be loaded is useful.

8.4.5 Image Compression

Almost all image compression reduces the refresh rate of the UI, so avoid using compressed images if the memory size is sufficient.

8.4.6 Font

Custom Binary Files

- Use multiples of 8 for font size whenever possible.
- When the file contains several hundred characters, *indexMethod* should be set to 0 when creating the font file.

Standard TTF Files

- Using TTF files to display text is significantly slower than using BIN.
- TTF files can be clipped through an open source solution.

8.5 Display

8.5.1 Font Anti-Aliasing

- Poor font anti-aliasing and abnormal colored edges on white text.

When using font libraries with 2 bits or more, if the font anti-aliasing effect is poor and there are abnormal colors on the edges of the font or the font color is displayed incorrectly, it may be an issue with the endianness of the font rendering data. To diagnose this, try displaying the font in RGB single-channel colors. For example, set the font color to `gui_rgb(255, 0,0,255)`. Normally, the text should appear red. If the text appears blue, there is an abnormality (this can also be identified with any colored text).

CHAPTER

NINE

GET PDF

PDF version: RTKIOT GUI.pdf

CHAPTER

TEN

GLOSSARY

API
Application Programming Interface

APP
Application

BG
Background

DMA
Direct Memory Access

FB
Frame Buffer

GPIO
General Purpose Input Output

GPU
Graphics Processing Unit

GUI
Graphical User Interface

I2C
Inter-Integrated Circuit

IC
Integrated Circuit

KB
Key Board

LCD
Liquid Crystal Display

OS
Operating System

PC
Personal Computer

PSRAM
Pseudo Static Random Access Memory

RAM
Random Access Memory

RLE
: Run-Length Encoding

RVD
: RTKIOT Visual Designer

TP
: Touch Pad

CHAPTER

ELEVEN

RELEASE NOTES

11.1 Major Changes

11.1.1 v1.0.6.6

- **Major Features**
  - Add view widget. (21a61e0d)
  - Add 3d face. (d2862e5e)
  - Add littlefs packing tool. (7bd59f3d)
- **Major Bug Fixes**
  - Fix roller loop off. (6045a92d)
  - Fix lv_rle, add fs load for rle, fix cache. (821b890e)

11.2 Change Logs

11.2.1 v1.0.6.6

- **Added**
  - Add view widget. (21a61e0d)
  - Add 3d face. (d2862e5e)
  - Add littlefs packing tool. (7bd59f3d)
- **Changed**
  - Modify LVGL watch demo. (92ed6cd8, e7cf2529, 93be85a9, 09f7cec0)
  - Modify LVGL doc. (eba1e59b)
- **Fixed**
  - Fix roller loop off. (6045a92d)
  - Fix lv_rle, add fs load for rle, fix cache. (821b890e)

INDEX

I

K

L

M

O

P

R

T

V