RTKIOT GUI Documentation

发行版本 *v0.0.0.1*

RTKIOT GUI community

2025 年 03 月 27 日

Contents

免责申明

本免责声明适用于由 Realtek 公司提供的所有文件，包括但不限于解决方案、用户指南、学习指导等（统称为“文件”）。这些文件按照“现状”提供，不对其完整性、精确性、及时性或适用性做任何明示或暗示的保证。用户同意仅出于约定目的使用文件，并遵守本免责声明的条款。

CHAPTER 1

入门指南

1.1 源代码项目下载

- 在 GitHub 上下载：https://github.com/realmcu/HoneyGUI
- 在 Gitee 上下载：https://gitee.com/realmcu/HoneyGUI

1.2 介绍

HoneyGUI 是由 Realtek 自主研发的图形显示框架，它是一款专为资源受限的微控制器和嵌入式系统设计的开源嵌入式图形用户界面（GUI）库。HoneyGUI 具备轻量级、功能丰富和高度可定制性的特点，广泛应用于消费电子、家电、医疗设备和智能手表等领域。

作为一套综合显示框架，HoneyGUI 不仅包含 Realtek 自主研发的显示引擎，还支持直接调用如 LVGL、ARM2D 等外部 API 进行应用程序的开发。此外，HoneyGUI 提供基于 PC 的仿真环境，使开发者能够快速进行应用开发和调试，而无需依赖嵌入式硬件平台。同时，HoneyGUI 还可以与 Realtek 自研的前端设计工具*RVD* 配合使用，实现可视化编程。

以下是几种常见的 APP 开发方式：

- 基于 RealGUI 显示引擎，调用 C/C++ API 开发应用程序。
- 也可以直接调用 LVGL 的 API 开发应用程序。
- 也可以直接调用 ARM-2D 的 API 开发应用程序。
- 前端开发方式，包含 JavaScript 和 XML。推荐使用 `RVisualDesigner` 作为*PC* 设计器进行低代码开发。

GUI 框架具有很强的可移植性，可以在多种芯片和*OS* 上运行。此次提供了 PC Windows 版本。

1.3 软件架构

1.4 安装在 Windows 操作系统上

1.4.1 安装编译器

下载 MinGW-w64 工具链，解压到 C 盘，并将其添加到系统环境变量 `Path` 中。

1. MinGW-w64 下载
2. 解压并复制到目录：`C:\mingw64`
3. 添加一个环境变量：`C:\mingw64\bin`：
   - 打开开始菜单，搜索 *高级系统设置*。
   - 显示 *系统属性*，然后转到 *高级*选项卡。
   - 点击 *环境变量*按钮。
   - 在 *用户变量*部分，找到并选择 `Path` 变量，然后点击 *编辑*。
   - 点击 *新建*并添加 `C:\mingw64\bin`。
   - 点击 *确定*关闭所有对话框。

1.4.2 安装 Python

测试过 Python 3.9.7 版本。

1.4.3 安装 Scons

打开一个 CMD 窗口，并执行以下命令来安装 Python 的 scons 库：

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

安装 `MinGW-w64` 工具链和 `scons` 库后，可以通过两种方式启动应用程序：通过 CMD 启动或通过 GUI 启动。

1.4.4 通过 CMD 启动（Scons）

在 `HoneyGUI` 或 `gui` 文件夹中打开一个 CMD 窗口，然后运行以下命令启动应用程序。

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

`scons` 命令执行构建过程，然后执行 `gui.exe` 来运行应用程序。

1.4.5 通过 CMD 启动（CMake）

- 依赖软件

  CMake（测试版本为 3.31.2）：https://cmake.org/download/

  MinGW-w64：如前所述

- 初始化：在 HoneyGUI 文件夹中

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

- 编译：在 HoneyGUI/win32_sim/build 文件夹中

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

- 配置：在 HoneyGUI/win32_sim/build 文件夹中

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

- 运行：在 HoneyGUI 文件夹中

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

1.4.6 通过 VSCode 启动

安装 VSCode

- 下载 VSCode
- 安装 C/C++ 插件

打开项目

- 单击 HoneyGUI.code-workspace 文件

运行项目

进入 VSCode 界面后，可以选择 Run and Debug 选项，然后点击 Run 按钮。

1.5 显示

1.5.1 手表工程

窗口中显示出表盘，您可以通过滑动和长按与其进行交互。

1.5.2 仪表盘工程

窗口中显示仪表盘。

CHAPTER 2

GUI 应用程序

HoneyGUI 框架的结构图如下所示：

图 1: HoneyGUI 框架

- 在每个项目中可以同时存在多个应用程序，但同一时间只有一个应用程序可以处于运行状态，其他应用程序将处于挂起状态。
- 使用不同的显示引擎时，应用程序的上层代码写法会有所不同。
- 每个*APP* 可以创建独立的线程，也可以选择不创建。
- APP 可以进行安装、打开、关闭、卸载和切换的操作。
- GUI_SERVER 根据刷新指令遍历控件，渲染帧缓冲区，执行触发回调，进行 APP 调度等操作。

2.1 C-APP 应用程序

- 在本章中，我们将探讨 GUI 框架内 C-APP 的创建与管理。C-APP 实质上是用户可以开发以制作互动和视觉吸引力用户界面的应用程序。每个 C-APP 都可以打开、关闭、切换，并在切换时应用动态过渡效果。
- C-APP 中显示的内容是使用嵌套控件树结构组织的。该结构包括诸如窗口、可滚动页面和可切屏的容器控件，以及诸如文本、图像和画布之类的内容显示控件。
- 除了默认功能和效果外，C-APP 内的控件提供了高度的自定义性。用户可以为控件设置自定义帧动画，并绑定事件以执行他们定义的操作。这种灵活性使得能够根据特定需求和要求创建高度动态和互动的用户界面。

2.1.1 定义一个 C-APP

- 使用特定名称通过 `GUI_APP_DEFINE_NAME_ANIMATION` API 定义应用程序句柄。
- 还有其他方式可以定义应用程序，例如：
  - `GUI_APP_DEFINE`
  - `GUI_APP_DEFINE_NAME`
  - `GUI_APP_DEFINE_NAME_ANIMATION_FUNC_CUSTOM`
  - `struct gui_app`
- 使用 `GUI_APP_ENTRY` API 定义应用程序的 UI 设计入口函数。
- 当应用程序启动时，UI 设计入口函数将被执行一次。

2.1.2 创建一个 C-APP 的控件树

- 这是一个时钟应用程序，作为本节的示例。
- 在下图中，您可以看到应用程序界面有秒表和倒计时器选项。
- 点击这些选项可以在它们之间切换。

下图显示了精简过的控件树结构：

- SCREEN:APP_STOPWATCH: 秒表应用程序的主容器。
  - WINDOW:LEFT_BUTTON: 包含左按钮的窗口。
    - CANVAS_RECT:LEFT_BUTTON: 左按钮的背景画布。
    - TEXTBOX:LEFT_BUTTON: 左按钮的文本标签。
  - WINDOW:RIGHT_BUTTON: 包含右按钮的窗口。
    - CANVAS_RECT:RIGHT_BUTTON: 右按钮的背景画布。
    - TEXTBOX:RIGHT_BUTTON: 右按钮的文本标签。
  - MULTI_LEVEL:0_0: 多级容器。
    - MULTI_LEVEL:1_0: 多级容器中的一个子容器，供秒表使用。
    - MULTI_LEVEL:1_1: 多级容器中的另一个子容器，供倒计时器使用。

2.1.3 C-APP 操作

以下是上述操作如何具体应用于秒表应用程序：

- `GUI_APP_SHUTDOWN(APP_STOPWATCH)`：此命令将关闭秒表应用程序。如果应用程序正在运行计时器，它将停止计时器并关闭界面。关闭时任何相关资源将被释放。
- `GUI_APP_STARTUP(APP_STOPWATCH)`：此命令将初始化并启动秒表应用程序。用户界面将会显示，应用程序将准备好开始记录时间。
- `GUI_APP_SWAP(APP_STOPWATCH, APP_MAP)`：这将在当前运行的秒表应用程序和地图应用程序之间切换。

2.1.4 C-APP 转场动画

C-APP 提供了一个强大的功能集来管理应用程序之间的转场动画。它主要提供三个功能：“内置动画”、“自定义动画”和“图层管理”。这些功能旨在通过提供流畅且视觉上令人愉悦的过渡来增强用户体验。

- 内置动画

  开发者可以使用 `GUI_APP_DEFINE_NAME_ANIMATION` API，轻松实现应用程序过渡的内置动画。此 API 允许您指定应用程序打开或关闭时发生的过渡动画。第二个参数用于定义应用程序启动时的动画，而第三个参数用于指定关闭应用程序的动画，如 `GUI_APP_ANIMATION_1`。这个简单的 API 简化了在应用程序中集成过渡效果的过程。

- 自定义动画

  对于更复杂或独特的动画需求，C-APP 支持通过 `GUI_APP_DEFINE_NAME_ANIMATION_FUNC_CUSTOM` API 来实现自定义动画。此功能允许开发者为应用打开和关闭过渡设置自定义动画回调函数。第二个参数是打开动画的回调函数，第三个参数是关闭动画的回调函数。这些回调函数通过 `GUI_ANIMATION_CALLBACK_FUNCTION_DEFINE` API 定义。这个 API 提供一个 `gui_animate_t` 实例作为参数，其中包含成员可以提供有关动画进度和状态的信息，从而允许进行精细的控制和定制。

- 图层管理

  C-APP 还包括用于管理应用程序图层的 API，这对于视觉层次结构和用户体验来说至关重要。`gui_app_layer_top` 和 `gui_app_layer_bottom` API 允许开发者定义当前活动应用程序与即将打开的应用程序之间的图层关系。此功能确保窗口的正确顺序，并有助于维护应用程序界面的预期焦点和组织。

示例

- 内置动画
  - 定义一个 C-APP
    - 启动动画：从屏幕中心放大 (`GUI_APP_ANIMATION_1`)
    - 关闭动画：缩小到屏幕中心 (`GUI_APP_ANIMATION_5`)
  - 切换到该 C-APP
    - 从应用程序 APP_WATCH 到 APP_STOPWATCH
- 自定义动画
  - 定义一个 C-APP
    - 启动动画：从屏幕底部弹出 (`heart_rate_startup`)
    - 关闭动画：滑动消失 (`heart_rate_shutdown`)
  - 切换到该 C-APP
    - 从应用程序 APP_WATCH 到 APP_HEART_RATE

2.1.5 API

警告： doxygenfile: Cannot find file “gui_app.h

2.2 使用 LVGL 设计应用程序

2.2.1 LVGL 简介

- LVGL 官网
- LVGL 在线文档
- LVGL 简介

LVGL（Light and Versatile Graphics Library）是最受欢迎的免费开源嵌入式图形库，可为任何 MCU、MPU 和显示类型创建漂亮的用户界面，提供了一切你需要创建嵌入式 GUI 所需的功能，包括易于使用的图形元素、美观的视觉效果和低内存占用。

LVGL 在其官网上展示了 Demo 效果以体现 LVGL 的 UI 构建能力。在线文档是 LVGL 的主要开发资料，其中详细介绍了 LVGL 的设计和运行逻辑、各个控件的使用方法、丰富的示例程序以及移植方法等。无论是新手还是有经验的开发者，都可以基于在线在线文档快速上手并深入理解 LVGL 的功能和特性。

- LVGL Demo
- LVGL Example

2.2.2 HoneyGUI 模拟器

模拟器是开发 UI 时使用的一个强大的工具，用于在计算机上模拟嵌入式设备的 UI 界面。它可以模拟真实硬件平台的行为和外观，提供给开发人员一个便捷的环境来快速创建、调试和测试 UI 设计。

模拟器的主要作用是实时展示和交互测试设计的 UI 界面，从而减少在实际硬件上进行反复测试的时间和成本。通过使用模拟器，开发人员可以快速迭代设计，实时查看效果，并进行调试和验证。这大大加快了 UI 的开发速度和质量，并提高了工作效率。

使用模拟器有以下优点：

- 实时预览：模拟器可以即时显示 UI 界面的效果，使开发人员能够快速看到设计的外观和功能效果，方便进行调整和修改。
- 跨平台支持：模拟器可以在计算机上运行，开发人员不需要依赖具体的硬件平台。
- 节省时间和资源：使用模拟器可以避免在实际硬件上频繁烧录和测试 UI，减少了额外的时间和成本开销。
- 调试和测试：模拟器提供了丰富的调试和测试功能，可以检查 UI 元素的交互、事件处理和布局效果，有助于解决问题和优化性能。

在 HoneyGUI 模拟器中运行 LVGL

HoneyGUI 模拟器基于 scons 工具和 MinGW-w64 工具链，在 VScode 中运行和进行调试，具体的环境配置和启动运行请参考入门指南 章节。

完成 HoneyGUI 模拟器的环境安装后，启动运行将看到模拟器默认的 HoneyGUI 工程。修改模拟器配置文件以运行 LVGL 的工程，在路径 `your HoneyGUI dir/win32_sim/` 下的 `menu_config.h` 文件为模拟器的配置文件，在 **HoneyGUI Demo Select** 下注释掉所有的 Demo，在 **HoneyGUI Enable LVGL** 下使能 `CONFIG_REALTEK_BUILD_LVGL_GUI`。在 VScode 中再次启动运行，构建编译通过后即可看到 LVGL 默认的 Demo 工程运行。

1. 当需要修改屏幕尺寸时，修改文件 `your HoneyGUI dir/realgui/example/demo/` 下的 `SConscript` 文件，修改其中的屏幕宽度 `DRV_LCD_WIDTH` 和屏幕高度 `DRV_LCD_HIGHT`，均为像素单位。

HoneyGUI LVGL

以下为 HoneyGUI 中与 LVGL 相关的目录及文件：

```
HoneyGUI Dir
|-- Arm2D
|-- cmake
|-- doc
|-- realgui
|   |-- 3rd
|   |-- app
|   |-- core
|   |-- dc
|   |-- engine
|   :
|   |__ example
|       |-- BAK
|       |-- demo
|       |   |__ app_ui_lvgl.c                // 模拟器 LVGL UI 入口
|       :
|       :
|       |__ screen_lvgl
|           |-- assets                       // LVGL 用户图片和字库 C 文件
|           |   |__ lvgl_example_assets.c    // assets example
|           |
|           |-- root                         // 文件系统根目录
|           |-- _bin_mkromfs.py
|           |-- mkromfs_0x4600000.bat        // User Data 打包脚本
|           |-- resource.h                   // 打包的文件资源地址映射
|           |__ root(0x4600000).bin          // 打包的 User Data
|
|-- keil_sim
|-- lib
|-- lvgl_v8                                  // LVGL v8.3
|   |-- demos                                // LVGL demo 源文件
|   |   |-- benchmark
|   |   |-- keypad_encoder
|   |   |-- music
|   |   |-- stress
|   |   |__ widgets
|   |
|   |-- docs
```

(续下页)

(接上页)

```
|   |-- env_support
|   |-- examples                              // LVGL example 源文件
|   |   |-- anim
|   |   |-- arduino
|   |   |-- assets
|   |   |-- event
|   |   |-- get_started
|   |   |-- layouts
|   |   |-- libs
|   |   |-- others
|   |   |-- porting                           // LVGL porting 模板
|   |   |-- scroll
|   |   |-- styles
|   |   |__ widgets                           // LVGL example 控件源文件，包含各控件␣
↪example
|   |
|   |-- rlottie
|   |-- scripts
|   |-- src
|   |   :
|   |   |-- widgets
|   |   |__ font                              // LVGL 内置字库
|   |
|   |__ tests
|
|-- lvgl_v9                                   // LVGL v9
|
:
:
|__ win32_sim
    :
    |__ port                                  // 模拟器 porting
        |-- realgui_port                      // 模拟器 HoneyGUI porting
        |-- lvgl_port                         // 模拟器 LVGLv8 porting
        |    |-- lv_conf.h                    // 模拟器 LVGL 配置定义
        |    |-- 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                        // 模拟器 LVGLv9 porting
```

1. HoneyGUI 中 LVGL 源文件在目录 `your HoneyGUI dir/lvgl` 下：

- demos：存放 LVGL 一些综合的内置示例，部分示例可以在 LVGL Demo 中体验。
- docs：存放 LVGL 的开发文档，可在 LVGL 的文档站点在线阅读：LVGL Document 。
- env_support：一些环境或者平台的支持。
- examples：存放 LVGL 的内置示例，可在 LVGL Example 中体验。
- scripts：存放一些处理脚本，在使用 LVGL 时基本不会用到。
- src：存放 LVGL 实际的源码，使用 LVGL 进行开发时，都是使用这里面的代码文件。
- tests：存放一些 CI 测试文件，在使用 LVGL 时不会用到。

2. HoneyGUI 模拟器运行 LVGL 时，LVGL UI 将从目录 `your HoneyGUI dir/realgui/example/demo` 下的 `app_ui_lvgl.c` 开始运行。
3. 使用 HoneyGUI 模拟器运行 LVGL 时，调用的 LVGL 文件系统接口所指向的根目录为 `your HoneyGUI dir/realgui/example/screen_lvgl/root/`。

2.2.3 实机移植

- 文档说明：LVGL Porting

LVGL 提供了广泛的移植支持，使开发者可以将其轻松地集成到各种嵌入式系统和平台中。它支持各种显示设备的驱动、触摸屏、输入设备和自定义 GPU 等。开发者可以根据项目的需求进行移植配置，例如更换显示设备时调整显示参数，替换输入设备时适配输入接口等。本文以显示设备、输入设备和文件系统为例，介绍移植过程和方法，更多细节请参考 LVGL Porting。

备注： 以下示例不包含硬件设备驱动的具体实现，仅示例如何将驱动对接到 LVGL 的接口。开发者在实现硬件设备驱动时，可在与示例驱动一致的 api 框架下来完成驱动功能，以对接到 HoneyGUI driver 层接口，往上则可复用示例工程的 porting 接口。

显示

- 文档说明：LVGL Porting Display, LVGL Overview Display

在开发者完成显示设备的驱动功能调试后，设备能够与显示设备正常通信并显示色彩。本小节介绍如何将驱动与 LVGL 的显示接口进行对接以展现 LVGL 的 UI 界面。

LVGL 的显示接口在文件 `lv_port_disp.c` 中实现，显示参数在初始化函数 `void lv_port_disp_init(void)()` 中进行配置，如屏幕尺寸和 frame buffer 配置准备等，显示刷新函数为 `void disp_flush(lv_disp_drv_t *disp_drv, const lv_area_t *area, lv_color_t *color_p)()`。

文件 `lv_port_disp.c` 中已配置好不同的绘制和推屏方式供参考，配置 `DISPLAY_FLUSH_TYPE` 以切换模式，其中 `RAMLESS_XXX` 适用于不带有 RAM 的 display IC，`RAM_XXX` 适用于带有 RAM 的 display IC，`XXX_FULL_SCREEN_XXX` 表示为每次整屏推出，`XXX_TWO_SEC` 表示为只绘制变化的显示内容，单位为两个 buffer 大小，buffer 的像素高度由 `SECTION_HEIGHT` 定义。

详尽的显示设备移植方法和注意事项请参阅文档 LVGL Porting Display，以下代码段示例了 porting 不带有 RAM 的 display IC：

- 使用不带有 RAM 的 display IC 时，必须为其分配整屏尺寸的 frame buffer，因此在 PSRAM 上分配了两个整屏尺寸的 frame buffer 用于显示。显示的参数宏定义已定义在文件 `lv_conf.h` 中。
- 若使用的 display IC 带有 RAM，则 frame buffer 的大小不必为整屏尺寸。由于刷屏方式的不同，需要配置 `lv_port_disp.c` 中的 `LVGL_USE_EDPI` 为不启用 (0)，以切换 `disp_flush()` 函数适配刷屏。

```
// 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
```

(续下页)

(接上页)

```
#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.
```

(续下页)

(接上页)

```
    *      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*/
```

(续下页)

(接上页)

```
    /*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);
}
```

输入设备

- 文档说明：LVGL Porting Input devices

在开发者完成输入设备的驱动功能调试后，设备能够与输入设备正常通信。本小节介绍如何将驱动与 LVGL 的输入接口进行对接以与 LVGL 的 UI 界面进行交互。

LVGL 的输入接口在文件 `lv_port_indev.c` 中实现，输入设备参数在初始化函数 `void lv_port_indev_init(void)()` 中进行配置，如选择设备类型等，输入数据获取函数配置在函数指针 `indev_drv.read_cb()`，取决于输入设备类型，均在 `lv_port_indev.c` 中对接。

详尽的输入设备移植方法和注意事项请参阅文档 LVGL Porting Input devices，以下代码段示例了 porting 触屏 IC：

- 在初始化函数 `void lv_port_indev_init(void)()` 中选择注册对应类型的输入设备，如触屏设备则选择 **Touchpad**
- LVGL 将通过函数指针 `indev_drv.read_cb()` 获取输入的数据，开发者需要在其指向的函数中提供输入数据，如触屏设备则为函数 `void touchpad_read(lv_indev_drv_t *indev_drv, lv_indev_data_t *data)()`。触屏输入设备仅需提供触点的坐标及触摸状态即可。

```
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;

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

(续下页)

(接上页)

```
    /*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;
}
/*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)
```

(续下页)

(接上页)

```
{
    /*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;
}
```

文件系统

- 文档说明：LVGL Overview File system

使用文件系统来管理存储介质使数据更加有条理和易于维护，可以提高外部存储设备的兼容性和跨平台性，通过文件系统接口，开发者可以方便地操作文件数据，更加灵活和高效。开发者对接文件系统到 LVGL 的文件系统接口，使资源数据与工程代码得以分开存储，缩短编译时间，提高开发效率，也增强了 UI 设计的灵活性。

LVGL 的文件系统接口在文件 `lv_port_fs.c` 中实现，文件系统在初始化函数 `void lv_port_fs_init(void)()` 中进行配置，包括文件系统的初始化、挂载盘符等，开发者需要将文件系统各功能的接口对接到对应的 LVGL fs porting 函数中，保证输入输出数据格式与接口定义的相一致。

详尽的文件系统移植方法和注意事项请参阅文档 LVGL Overview File system，以下示例了 **ROMFS** porting 的部分接口。

备注： ROMFS 是一个只读文件系统，故不支持文件写入。

```
#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();

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

    /*Add a simple drive to open images*/
```

(续下页)

(接上页)

```
    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*/
    }
    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*/
    }
```

(续下页)

(接上页)

```
    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 文件系统镜像

HoneyGUI 提供 *ROMFS* 文件系统镜像的打包支持：

1. 工作路径为 `your HoneyGUI dir/realgui/example/screen_lvgl/`，执行打包过程需要有 python 环境支持，工程用到的外部文件资源将打包为文件系统镜像最终作为 *User Data* 下载。
2. 打开工作路径，将需要打包的文件放置于 `root/` 文件夹下，双击脚本 `mkromfs_0x4600000.bat` 生成文件系统镜像 `root(0x4600000).bin` 和资源映射地址 `resource.h`。文件的默认 *base address* 为 *0x4600000*，`resource.h` 中记录了打包文件的映射地址，由于 *ROMFS* 支持物理地址直接访问，开发者可通过映射地址直接访问资源文件。
3. 请使用 MP Tool 的 *User Data* 功能下载烧录文件系统镜像到 flash，烧录地址需与 *base address* 保持一致。

若需要修改 *base address*, 修改脚本 `mkromfs_0x4600000.bat` 中的 **"–addr <number>"** 参数即可，如下示例为修改 *base address* 从 *0x4600000* 改为 *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
```

备注:

1. 该打包工具仅适用于 ROMFS 的文件系统镜像打包。
2. 打包过程并非简单的文件拼接，同时也记录了文件系统的目录信息和文件的信息。

LittleFS 文件系统镜像

LittleFS 文件系统支持读写操作，且具有掉电保护的特点，HoneyGUI 提供 *LittleFS* 文件系统镜像的打包支持：

1. 工作路径为 `your HoneyGUI dir/realgui/example/screen_lvgl/root_lfs`，工程用到的外部文件资源将打包为文件系统镜像最终作为 *User Data* 下载。
2. 打开工作路径，将需要打包的文件放置于 `root/` 文件夹下，双击脚本 `mklittlefs_img.bat` 生成文件系统镜像 `root.bin`。
3. 请使用 MP Tool 的 *User Data* 功能下载烧录文件系统镜像到 flash。若需要修改文件系统的大小, 修改脚本 `mklittlefs_img.bat` 中的 **"-s <number>"** 参数即可。当使用 `rtk_fs.c` 中的接口进行文件操作时，其中的 `RTK_FS_MNT_ADDR` 需与烧录地址一致，`MAX_LFS_SIZE` 需与文件系统大小一致。
4. 如需解包文件系统镜像，双击脚本 `unpack_littlefs_img.bat` 将 `root.bin` 解包到 `root_up/` 文件夹下。

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

(续下页)

(接上页)

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

备注:

1. 该打包工具仅适用于 LittleFS 的文件系统镜像打包。

2.2.4 LVGL Benchmark 测试

LVGL 的 Benchmark 是一个性能测试工具，用于评估 LVGL 库在各种硬件和软件环境下的图形显示性能。通过运行 Benchmark，用户可以获取帧率、渲染速度和内存使用情况等数据，从而帮助优化显示配置和调试性能问题。Benchmark 包括多种测试场景，如图形绘制、动画和文本渲染，每个场景模拟实际应用中的常见操作。用户可以通过这些测试来比较不同配置和平台的性能表现，从而做出针对性的优化调整。LVGL 基准测试的官方文档位于 `your HoneyGUI dir/lvgl/demos/benchmark/README.md`。

参考 Benchmark

表 1: Benchmark 测试结果

芯片型号	处理器主频	加速器	显示面积	缓冲区配置	结果
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%

表 2: 不同平台渲染加速

芯片型号	处理器主频	硬件加速器	图片绘制	图片透明度	图片缩放	图片旋转	圆角矩形	矩形填充	RLE 解码	字符	线条
RTL8772	125MHz	PPE1.0	HW	HW	HW	SW	SW+HW	HW	HW	SW	SW
RTL8773	100MHz	PPE2.0	HW	HW	HW	HW	SW+HW	HW	HW	SW	SW

备注:

1. 涉及 LVGL Mask 的效果均需要 SW 处理
2. RTL8772G 支持 Helium 硬件加速器

2.2.5 从 Demo 入门开发

- LVGL Demo
- LVGL Example

建议开发者开发前先行阅读理解 LVGL Overview 和 LVGL Widgets - Base object 部分以了解 LVGL 的设计概念和设计逻辑。

LVGL 提供了丰富的 demo 和 example 来帮助开发者了解熟悉各个控件和特性的使用。

- LVGL Demo 中展示了综合性比较强的 Demo，其源码保存在目录 `your HoneyGUI dir/lvgl/src/demo` 下，开发者可直接调用对应的 `lv_demo_xxx()` 函数来熟悉了解。
- 在线文档 LVGL Example 中展示了各个 example 的运行效果，其源码保存在目录 `your HoneyGUI dir/lvgl/src/example` 下，开发者可直接调用对应的 `lv_example_xxx()` 函数来熟悉控件和理解特性。

2.2.6 资源转换器

LVGL 的图片和字库需要借助工具转换为 LVGL 可以识别的格式，才能在 UI 中使用。LVGL 支持转换为 C 数组格式和 bin 二进制文件的资源，其中 C 数组格式的资源将会参与编译过程，每当程序逻辑发生变化时，都会参与编译，资源大小计入 APP image（OTA 时需要更大空间），bin 二进制文件格式的资源不参与编译，单独存储，需要文件系统等来支持访问。在路径 `your HoneyGUI dir/realgui/example/screen_lvgl/assets/` 下已提供 example `lvgl_example_assets.c` 示例如何为控件配置不同格式的资源。

图片转换器

LVGL 在线转换工具

- 在线转换工具：LVGL Image Converter
- 文档说明：LVGL Overview Images

使用步骤请参考 LVGL Overview Images - Online Converter：

1. 选择 LVGL 版本
2. 选取图片文件
3. 选择输出文件的颜色格式

   颜色格式的说明请参考 LVGL Overview Images - color format
4. 选择输出图片的类型 (C array/binary file)
5. 点击 *Convert* 获取输出文件

在文档 LVGL Overview Images 中详细介绍了如何在 LVGL 中使用图片资源和图片转换工具，并提供了简单的使用范例。以 C array 生成的图片资源置于 `your HoneyGUI dir/realgui/example/screen_lvgl/assets/` 下即可被自动构建到工程中。

值得一提的是，使用 bin 文件的图片资源时，bin 文件中数据的格式为 `4 Byte lv_img_header_t + data`，其中 `lv_img_header_t` 中包含有 `Color format`，`width` 和 `height`，此时利用 `lv_img_header_t` 信息来计算出 `data_size` 即可构建一个完整的 `lv_img_dsc_t` 来描述图片。

```
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*/

    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 图像转换工具

- 转换工具下载链接：HoneyGUI Image Convert Tool
- 文档说明：HoneyGUI Image Convert Tool - Doc

当需要进一步压缩图片资源占用空间时，HoneyGUI 图像转换工具支持对图片进行压缩转换，IC 支持软硬件解码。HoneyGUI 图像转换工具采用 RLE(Run-length Encoding) 压缩，该压缩算法是一种简单的无损算法，通过编码连续重复的像素值和重复次数来减少存储空间，计算复杂度低且压缩率较高，非常适合用于压缩 GUI 资源。

压缩图片

用户可利用 HoneyGUI 图像转换工具将图片资源转换为 RLE 压缩的二进制文件格式，具体使用步骤请参考 HoneyGUI Image Converter - Doc：

1. 选择需要压缩的图片文件（支持 PNG、JPEG 等格式）
2. 配置图片的转换参数：启用 *Compress*，*Compress Mode* 选择 *RLE*，启用 *Color Head*，*Color Space* 按需选择
3. 点击 *Convert* 生成压缩的二进制文件

导入 LVGL

HoneyGUI 图像转换工具生成的二进制文件可导入 LVGL 使用：

1. 若作为文件导入

   注意: 修改文件扩展名为 **.rle** , 即可放入文件系统使用 `your HoneyGUI dir/realgui/example/screen_lvgl/root`

   ```
   // file: lvgl_example_assets.c
   void load_img_rle_file(void)
   ```

(续下页)

(接上页)

```
{
    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);
}
```

备注：使用 RLE 解码器 + ROMFS 时，解码器将会直接从文件系统即 FLASH 上获取图片，不做额外缓存，需要做缓存处理的情况请使用文件系统接口将文件读到内存中，作为数组方式使用。

2. 若作为 C 数组格式导入

- a. 打开 LVGL 图片转换在线工具并上传要转换的压缩文件，请参考*LVGL 在线转换工具*
- b. 在 *Color format* 选项中，务必选择 **CF_RAW**
- c. 将转换后的图片文件导出为 C 文件格式，例如 `logo_lvgl_rle.c`

  注意 1：转换结果文件的存放路径: 将转换后的 C 文件存放在以下参考路径：`your HoneyGUI dir/realgui/example/screen_lvgl/assets`

  注意 2：修改图像描述符中的色彩格式 cf: 导出的 C 文件，例如 `logo_lvgl_rle.c`，需要对其中的图像描述符进行修改，保证 *cf* 设置为 `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. 在项目中声明图片后即可作为图片源使用

```
// 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. 若作为文件导入，以文件地址的方式访问图片资源

- a. 构建 `lv_img_dsc_t` ，例如：

```
// 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,
```

(续下页)

(接上页)

```
    .data = LOGO_LVGL_RLE,
};
```

注意：图像描述符中的色彩格式设置为 cf = LV_IMG_CF_RAW

- b. 图片资源访问，控件创建：

```
// 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);
}
```

LVGL 启用 RLE 解码器

为了在 LVGL 中解码 RLE 压缩的图片资源，需要配置启用 RLE 解码器，并为其分配缓存空间。

1. 启用 RLE 解码器：在配置文件 `lv_conf.h` 中找到 `LV_USE_RTK_IDU` 宏定义，并将其设置为启用 (1)
2. **分配解码缓存：在 `lv_conf.h` 文件中配置以下参数：**
    - `LV_PSRAM_START`：缓存的起始地址
    - `LV_PSRAM_SIZE`：缓存空间大小，确保此大小足够容纳所使用的最大整张图片的解码数据

```
// 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
```

备注： 使用 RLE 解码器 + ROMFS 时，解码器将会直接从文件系统即 FLASH 上获取图片，不做额外缓存；

字库转换器

- 在线转换工具：LVGL Font Converter
- 文档说明：LVGL Overview Fonts

使用步骤请参考 LVGL Overview Font - Add a new font：

1. 设定输出字库的名字
2. 设定字体的高度 height，像素单位
3. 设定字体的 bpp(bit-per-piel)

   表示采用多少个 bit 来描述一个像素，当数值越大时，字符的抗锯齿效果越好，边缘越平滑，字库占用空间越大
4. 选择输出字库的类型 (C array/bin file)
5. 选择字体文件 (TTF/WOFF)
6. 设定需要转换的字符 Unicode 范围，也可直接列出需要转换的字符

在文档 LVGL Overview Fonts 中详细介绍了如何在 LVGL 中使用字库资源和字库转换工具，并提供了简单的使用范例。在 example 中 `lv_example_label_3()` 示例了如何为 label 控件配置指定的字库。以 C array 生成的字库资源置于 `your HoneyGUI dir/realgui/example/screen_lvgl/assets/` 下即可被自动构建到工程中。

在 LVGL 中提供了内置的字库，以数组的形式保存在目录 `your HoneyGUI dir/lvgl/src/font/` 下，每份字库所包含的字符均注明在文件开头。内置字库中包含有一份汉字字库 `lv_font_simsun_16_cjk.c` cjk 16 号字库，但为单一字号，字符数有限。

2.2.7 开发资源支持

在线文档

- LVGL Document

LVGL 的 在线文档 提供了全面的技术文档和教程，帮助开发者更好地了解和使用 LVGL 图形库。该文档包含以下内容：- 概述和特性：文档介绍了 LVGL 的基本概念和特性，包括图形对象、屏幕管理、事件处理、主题样式等。用户可以通过阅读文档了解 LVGL 的核心功能和优势。

- 应用开发指南：文档提供了详细的应用开发指南，包括如何初始化和配置 LVGL 、如何创建和管理图形对象、如何处理用户输入和事件、如何添加主题和样式等。这些指南可以帮助用户快速上手使用 LVGL 并开发自己的应用程序。
- API 文档：文档详细列举了 LVGL 的 API 接口和函数，以及它们的参数和用法。用户可以根据需要查阅 API 文档来了解具体的函数和接口的功能和用法，以便进行更高级的自定义和扩展。
- 示例代码：文档中提供了众多的示例代码，涵盖了常见的应用场景和功能。用户可以借鉴这些示例代码，加快开发速度，并快速实现特定功能的需求。

使用 LVGL 的在线文档可以帮助用户更好地理解和掌握 LVGL 的使用方法和技巧，提高开发效率。用户可以通过逐步学习文档中的内容，从简单的界面构建到复杂的应用开发，逐步掌握 LVGL 的各种功能和特性。同时，文档还提供了示例和代码片段，方便用户更快地开发出具有丰富界面和功能的应用程序。

用户可以通过在网页浏览器中打开 LVGL 的在线文档，并浏览各个章节和内容，根据自己的需要查找和学习相关的知识。另外，用户还可以通过搜索功能来快速查找文档中的具体信息。总之，LVGL 的在线文档是用户理解和使用 LVGL 图形库的重要资源，可以提供全面而详细的指导，帮助用户快速上手和开发出更好的应用程序。

基于文档开发能够完成大部分的 UI 效果，值得注意的是，文档内容并不一定齐全，当文档内容存在疏漏时，最终还是以代码为准。

Github 仓库

- Github LVGL

LVGL 的 GitHub 仓库是开发者使用和贡献 LVGL 的重要平台：

- 获取最新版本：LVGL 的 GitHub 仓库可以获得最新的 LVGL 版本和更新。开发者可以及时获取最新的功能更新、修复和改进，保持应用程序与 LVGL 的同步。
- 参与社区和贡献代码：通过 GitHub 仓库，开发者可以积极参与 LVGL 社区的讨论和交流，了解其他开发者的问题和解决方案。同时，开发者也可以贡献自己的代码和改进，让 LVGL 更加完善和强大。
- 提交问题和 bug 报告：GitHub 仓库提供了问题和 bug 报告的平台，开发者可以提交他们在使用 LVGL 过程中遇到的问题和 bug。这有助于 LVGL 开发团队及时发现和解决问题，提高 LVGL 的稳定性和可靠性。
- 学习示例和文档：GitHub 仓库中还包含示例代码和文档，帮助开发者更好地理解和学习 LVGL 的使用。开发者可以通过浏览仓库中的示例代码和文档，学习 LVGL 的各个功能和特性，提高开发技能。

设计器

- GUI Guider: 免费
- Squareline: Squareline Studio，付费

LVGL 的设计器（LVGL Designer）是一个为 LVGL 图形库设计和开发界面的可视化工具。它提供了一个直观且用户友好的界面，使开发者能够快速创建和编辑 LVGL 的 GUI 界面。

LVGL Designer 具有以下特点和功能：

- 可视化界面设计：设计器提供了直观的可视化界面，开发者可以使用鼠标和简单的拖放操作来创建和编辑 GUI 界面。它允许添加和调整各种图形对象、标签、按钮、文本框、图像等元素，并设置它们的大小、位置、样式等属性。
- 实时预览和调试：设计器支持实时预览，开发者可以随时查看他们所设计的界面的外观和行为。这有助于开发者快速调整和优化界面，确保其满足预期效果。
- 事件和交互管理：设计器使开发者能够方便地添加和管理事件和交互行为。开发者可以为图形对象添加点击、滚动、拖动等事件，并通过简单的配置设置它们的响应行为。
- 主题和样式定制：设计器支持主题和样式的定制，开发者可以轻松地选择和应用不同的主题和样式，使界面更具个性化和美观。
- 导出代码：设计器允许将设计的界面导出为 LVGL 代码，并提供所需的初始化和配置。这样，开发者可以直接将导出的代码用于 LVGL 应用程序的开发，省去手动编写代码的步骤。

使用 LVGL 的设计器可以极大地加速 GUI 界面的设计和开发过程，尤其适用于非专业的 UI 设计师或开发者。通过简单的拖放和配置操作，开发者可以快速创建出具有吸引力和交互性的界面，提高开发效率和用户体验。同时，设计器还提供一个便捷的方法来导出设计的界面为可用的 LVGL 代码，使开发者能够直接将其集成到他们的应用程序中。

论坛

- LVGL Forum

LVGL 的官方论坛是一个开发者社区，致力于讨论和分享有关 LVGL 图形库的话题和资源。它提供了一个平台，供开发者之间交流、寻求帮助和分享他们的经验和项目。

LVGL 论坛的一些特点和功能包括：

- 提问和回答：开发者可以在论坛上提出他们在使用 LVGL 时遇到的问题，并获得其他开发者的帮助和回答。这使得论坛成为一个宝贵的知识库，提供了解决问题的经验和技巧。
- 教程和示例：论坛上有许多有用的教程和示例代码，展示了如何使用 LVGL 的不同功能和特性。这些资源对于新手开发者学习和掌握 LVGL 非常有帮助。
- 开发者贡献和项目展示：论坛上的开发者可以分享他们的项目和定制的 LVGL 界面，以及其他开发者可以共享、讨论和参考的贡献。
- 更新和发布通告：LVGL 的开发团队在论坛上发布关于新版本发布和更新的通告和说明。这使得开发者可以及时了解最新功能和改进。
- 社区互动：论坛提供了一个社区互动的平台，开发者可以互相交流、分享和建立联系，加强 LVGL 开发社区的合作和发展。

LVGL 论坛对于使用 LVGL 的开发者来说，是获取支持、解决问题、学习和分享经验的重要资源。

博客

- LVGL Blog

LVGL 的官方博客是一个定期更新的平台，提供关于 LVGL 图形库的最新信息、教程、案例研究和开发者见解。LVGL 的开发团队和社区成员经常在博客上发布有关 LVGL 的各种内容，这些内容可以使开发者更好地了解和使用 LVGL。

LVGL 的博客包含以下内容：

- 更新和新功能介绍：博客上会发布关于 LVGL 最新版本的更新和改进的文章，这些文章介绍了新功能、修复了的问题和性能提升，使开发者可以了解和利用最新的 LVGL 特性。
- 教程和使用指南：博客会提供有关 LVGL 的实用教程和使用指南，涵盖从入门到高级的各种主题。这些教程通常包括示例代码和详细说明，帮助开发者掌握 LVGL 的使用和最佳实践。
- 案例研究和项目展示：博客上会分享一些使用 LVGL 实现的案例研究和项目展示。这些文章介绍了如何使用 LVGL 构建实际应用程序和界面，让开发者从实践中获取灵感和经验。
- 技术深入解析和开发者见解：博客还会涵盖对 LVGL 的深入分析和开发者的见解。这些文章可能探讨 LVGL 的内部工作原理、性能优化技巧、优秀设计实践等方面的主题，给开发者提供更深入的了解和思考。

LVGL 的博客是一个重要的资源，对于 LVGL 的开发者来说是了解和掌握 LVGL 的宝贵来源。通过阅读博客，开发者可以获取到关于 LVGL 最新动态、学习材料和技术见解，帮助他们更好地使用 LVGL 构建出优秀的图形界面。

2.2.8 常见问题

- LVGL FAQ

HoneyGUI vs LVGL 绘制图片帧率

GRAM 屏幕 (280x456)SRAM 分块绘制

背景：RTL8772G 平台，RGB565，非压缩图片，测试单张图片的显示绘制性能。

表 3: RAM 分块绘制测试结果

测试类型	HoneyGUI 帧率 (FPS) SW	HoneyGUI 帧率 (FPS) PPE	LVGL 帧率 (FPS) SW	LVGL 帧率 (FPS) PPE
绘制图片	73	74	70	73
普通填充矩形	3	85	74	74
图像旋转 45°	3	3	4	4
图像放大 1.5 倍	3	31	3	25
图像缩小 0.5 倍	9	73	12	59

表 4: RAM 分块绘制测试数据

Section	HoneyGUI 帧率 (FPS)	LVGL 帧率 (FPS)
10	70	45
20	73	73
30	74	73

PSRAM 整帧 buffer 绘制（800x480）

背景：RTL8772G 平台，rgb565，图片尺寸 315x316，非压缩图片，RGB 屏幕，测试单张图片的显示绘制性能。

表 5: PSRAM 整帧 buffer 绘制

测试类型	HoneyGUI SW (FPS)	HoneyGUI PPE (FPS)	LVGL SW (FPS)	LVGL PPE (FPS)
绘制图片	76	76	17	25
普通填充矩形	4	78	25	26
图像旋转 45°	3	3	6	4
图像放大 1.5 倍	2	23	3	13
图像缩小 0.5 倍	10	82	13	50

分析

对于 RGB 屏幕需要额外的 psram 作为缓存 buffer，LVGL 完全使用 psram 作为图像缓存 buffer，相比于 HoneyGUI 采用 ram 与 sram 结合的方式，LVGL 各方面性能表现较差；

HoneyGUI vs LVGL RAM 消耗

表 6: GRAM 屏幕 (280x456) 动态 RAM 消耗

测试类型	HoneyGUI (Bytes)	LVGL 控件消耗 (Bytes)
绘制图片	156	176
普通填充矩形	64	200
图像旋转	156	208
图像放大 1.5 倍	156	208
图像缩小 0.5 倍	156	176

表 7: GRAM 屏幕 (280x456) 静态 RAM 消耗

测试类型	HoneyGUI (Bytes)	LVGL 控件消耗 (Bytes)
绘制图片	41892(40KB)	55300(54KB)
普通填充矩形	41892(40KB)	55300(54KB)
图像旋转	41892(40KB)	55300(54KB)
图像放大 1.5 倍	41892(40KB)	55300(54KB)
图像缩小 0.5 倍	41892(40KB)	55300(54KB)

结论

- **适用场景**: 需要推动大尺寸的屏幕（例如 800x480），并且整帧绘制的情况，推荐选择 HoneyGUI，对于需要频繁刷新脏块的项目，推荐使用 LVGL；分块绘制场景，在 ram 资源紧张的情况下，推荐使用 HoneyGUI，section 推荐参数 10。
- **旋转，放大缩小**：LVGL 在图像旋转方面由于采用 2x2 的矩阵，在二维图渲染方面，相比于 HoneyGUI 的 3x3 矩阵，运算方面数据量更少，因此表现更快，而对于显示 2.5D，仿三维效果时，HoneyGUI 将表现更好。
- 在实际项目中，可以根据具体的帧率需求、系统资源情况以及其他功能需求，选择合适的显示框架。如果可行，进行具体的性能测试和评估是最为理想的做法。

通过以上分析，可以为项目选择显示框架时提供参考，帮助决策人员根据实际需要做出最佳选择。

2.3 使用 ARM-2D 设计应用程序

Arm-2D 是一个在 Cortex-M 处理器上进行 2.5D 图像处理的开源项目。

- 初始目标：物联网终端设备、白色家电、手持设备和可穿戴设备，特别是对于资源受限和低功耗需求的设备。
- 初始重点：图形用户界面（GUI）开发。

2.3.1 ARM-2D 介绍

- ARM-2D

2.4 使用可视化工具设计应用程序

2.4.1 概述

RTKIOT 可视化设计工具是一个用于为 Realtek 系列 IC 制作图形界面设计的工具，目前支持的 IC 列表如下表所示。

表 8: 支持的 IC

序号	支持的 IC
1	RTL8762D
2	RTL8762G
3	RTL8763E
4	RTL8772G
5	TBD

RTKIOT 可视化设计工具支持以下功能：

- 从工具箱中拖拽控件并将其放置在设计视图中；
- 拖放控件以在设计视图中更改其位置，或通过属性视图修改控件的位置和外观；
- 导出用户设计的 GUI 项目为 `.bin` 文件，将 `.bin` 文件烧录到 IC 上以显示图形界面；
- 在 PC 上模拟 GUI 项目。

本文档主要包括以下内容：

- 功能面板
- 资源管理
- 菜单栏
- 快速入门教程
- *GUI* 演示项目

为了简化文档，下文中使用 工具来指代 *RTKIOT* 可视化设计工具。

2.4.2 功能面板

工具箱/控件

- 非容器化控件
  - 可作为其他控件的父控件。
  - 父控件与子控件之间存在坐标跟随关系。
  - 当子控件超出父控件范围时仍可显示。
- 容器化控件

- 可作为其他控件的父控件。
- 父控件与子控件之间存在坐标跟随关系。
- 当子控件超出父控件范围时仍可显示。
- 可从工具箱中将控件拖放到容器控件中。

本节列出了小部件支持的属性，并用 **Y** 或 **N** 标记是否 IC 支持该属性。

非容器化控件

文本（Text）

仅用于文本显示，不支持用户输入。属性如下表所示。

表 9: 文本（Text）控件属性

属性	描述	8762I	8762C	TBD:G
Name	控件名称	Y	Y	Y
Size (Height)	控件高度	Y	Y	Y
Size (Width)	控件宽度	Y	Y	Y
X	相对于父控件的水平坐标	Y	Y	Y
Y	相对于父控件的垂直坐标	Y	Y	Y
Text	显示文本	Y	Y	Y
Dis-play Mode	长文本（超出控件范围的文本内容）的显示模式可使用以下支持的类型： 截断显示模式（truncate）：截断文本显示模式 垂直滚动显示模式（verticalscroll）：垂直滚动文本显示模式 水平滚动显示模式（horizontalscroll）：水平滚动文本显示模式	Y	Y	Y
Font Font	字体设置请参考：字体转换设置	Y	Y	Y
Font Color (RGBA	字体颜色设置，使用 RGBA	Y	Y	Y

按钮（Button）

可点击的控件，具有文本和背景图片。属性如下表所示。

表 10: 按钮（Button）控件属性

属性	描述	8762D	8762G	TBD
Name	控件名称	Y	Y	Y
Size (Height)	控件高度	Y	Y	Y
Size (Width)	控件宽度	Y	Y	Y
X	相对于父控件的水平坐标	Y	Y	Y
Y	相对于父控件的垂直坐标	Y	Y	Y
Text	显示文本	Y	Y	Y
Text X	相对于按钮控件的水平坐标	Y	Y	Y
Text Y	相对于按钮控件的垂直坐标	Y	Y	Y
Display Mode	水平或垂直显示	Y	Y	Y
Font Font	字体设置请参考：字体转换设置	Y	Y	Y
Text Color (RGB)	文本颜色设置，使用 RGB	Y	Y	Y
Transition	图像转换模式有以下选项： normal: 无效果 fade: 淡入/淡出 scale: 缩放 fadeScale: 淡入/淡出和缩放 注意：只有在设置了默认和高亮背景图片时，变换模式才会生效，否则默认为 normal	N	Y	Y
BG Image (Default)	默认的背景图像	Y	Y	Y
BG Image (Highlight)	选定/高亮显示的背景图像	Y	Y	Y
BG Imge Rotation Angle	背景图像旋转角度，范围：0~360 度	Y	Y	Y

图像（Image）

能够设置图像的控件，其属性如下表所示。

表 11: 图像（Image）控件属性

属性	描述	8762D/8	8762G/8	TBD
Name	控件名称	Y	Y	Y
Size (Height)	控件高度	Y	Y	Y
Size (Width)	控件宽度	Y	Y	Y
X	相对于父控件的水平坐标	Y	Y	Y
Y	相对于父控件的垂直坐标	Y	Y	Y
Image	图像路径 注意：图像必须预先导入到项目中。详细请参考图像资源管理	Y	Y	Y
Image Rotation Angle	图像旋转角度	Y	Y	Y
Image Scale X	图像水平缩放程度，是一个倍数/百分比。 例如，设置比例 x 为 0.5 表示图像的实际显示宽度是原始图像宽度的一半	Y	Y	Y
Image Scale Y	图像垂直缩放程度，是一个倍数/百分比	Y	Y	Y

备注:

1. 在导出时，工具将转换导入的图像。可以在 菜单栏 ▸ 设置 ▸ 图像转换设置中设置图像转换参数，详细请参考图像转换设置;
2. 如果导入的图像大小与控件的大小不匹配，工具不会对图像进行缩放或裁剪。

滑动条（SeekBar）

滑动控件，可以响应用户滑动手势，并改变进度值。其属性如下表所示。

图 2: 图像滑动条（SeekBar）

表 12: 滑动条（SeekBar）控件属性

属性	描述	8762D/87	8762G/87	TBD
Name	控件名	Y	Y	Y
Size (Height)	控件高度	Y	Y	Y
Size (Width)	控件宽度	Y	Y	Y
X	相对于父控件的水平坐标	Y	Y	Y
Y	相对于父控件的垂直坐标	Y	Y	Y
Color(Highlight) (RGBA)	进度条完成部分的背景色	N	Y	N
Color (RGBA)	整个进度条的背景色	N	Y	N
Orientation	控件显示方向和手势响应方向的类型如下： vertical/V: 垂直方向 arc: 曲线的方向 horizontal/H: 水平方向	Y	Y	Y

图像滑动条（Image SeekBar）

具有多个图像作为背景的滑动控件，用户滑动时可以切换到不同的图像，其属性如下表所示。

表 13: 图像滑动条（ImageSeekBar）控件属性

属性	描述	8762D/8763E	8762G/8772G	TBD
Name	控件名	Y	Y	Y
Size (Height)	控件高度	Y	Y	Y
Size (Width)	控件宽度	Y	Y	Y
X	相对于父控件的水平坐标	Y	Y	Y
Y	相对于父控件的垂直坐标	Y	Y	Y
Degree (Start)	起始角度（如果方向是曲线则无效）	Y	Y	Y
Degree (End)	结束角度（如果方向是曲线则无效）	Y	Y	Y
Image Directory	包含要在此控件上显示的图像的文件夹 注意: 1. 请按名称对图像进行排序; 2. 当用户在控件上滑动时，控件将根据当前进度切换背景图像。	Y	Y	Y
Central X	弧的中心相对于父控件的水平坐标	Y	Y	Y
Central Y	弧的中心相对于父控件的垂直坐标	Y	Y	Y
Orientation	控件显示方向和手势响应方向的类型如下: vertical/V: 垂直方向 arc: 曲线的方向 horizontal/H: 水平方向	Y	Y	Y

开关（Switch）

具有 **已选中**和 **未选中**状态的开关控件，其属性如下表所示。

表 14: 开关（switch）控件属性

属性	描述	8762D/8763E	8762G/8772G	TBD
Name	控件名称	Y	Y	Y
Size (Height)	控件高度	Y	Y	Y
Size (Width)	控件宽度	Y	Y	Y
X	相对于父控件的水平坐标	Y	Y	Y
Y	相对于父控件的垂直坐标	Y	Y	Y
BG Image (Checked)	已选中状态的背景图像	Y	Y	Y
BG Image (Default)	未选中状态的背景图像	Y	Y	Y

圆弧（Arc）

弧形控件，暂时不支持手势，其属性如下表所示。

表 15: 圆弧（Arc）控件属性

属性	描述	8762D/8763E	8762G/8772G	TBD
Name	控件名称	Y	Y	N
Size (Height)	控件高度	Y	Y	N
Size (Width)	控件宽度	Y	Y	N
X	相对于父控件的水平坐标	Y	Y	N
Y	相对于父控件的垂直坐标	Y	Y	N
Central X	圆弧的中心相对于父控件的水平坐标	N	Y	N
Central Y	圆弧的中心相对于父控件的垂直坐标	N	Y	N
BG Color	圆弧背景颜色	N	Y	N
Cap Mode	圆弧端点样式，支持以下选项： 圆形/平头/方形	N	Y	N
Degree (End)	圆弧的结束度数	N	Y	N
Degree (Start)	圆弧的开始度数	N	Y	N
Radius	圆弧的半径	N	Y	N
Stroke Width	圆弧的描边宽度	N	Y	N

容器控件

屏幕（Screen）

屏幕控件，对应于物理屏幕，是 GUI 项目的根控件，其属性如下表所示。

表 16: 屏幕（Screen）控件属性

属性	描述	8762D/8763E	8762G/8772G	TBD
Name	控件名称	Y	Y	Y
Size (Height)	控件高度	Y	Y	Y
Size (Width)	控件宽度	Y	Y	Y
X	相对于父控件的水平坐标	Y	Y	Y
Y	相对于父控件的垂直坐标	Y	Y	Y

备注： 只能修改 Name 属性。

选项卡视图（TabView）和选项卡（Tab）

选项卡视图控件作为父控件，支持上下左右滑动来在选项卡之间切换。选项卡视图控件具有以下属性，如下表所示。

图 3: 选项卡视图（TabView）和选项卡（Tab）

表 17: 选项卡视图（TabView）属性

属性	描述	8762D/87(	8762G/87]	TBD
Name	控件名称	Y	Y	Y
Size (Height)	控件高度	Y	Y	Y
Size (Width)	控件宽度	Y	Y	Y
X	相对于父控件的水平坐标	Y	Y	Y
Y	相对于父控件的垂直坐标	Y	Y	Y
Transition	选项卡切换模式支持以下类型： normal: 无特效 fade: 淡入/淡出 scale: 缩放 fadeScale: 淡入/淡出和缩放	N	Y	Y

表 18: 选项卡（Tab）属性

属性	描述	8762D/8763E	8762G/8772G	TBD
Name	控件名称	Y	Y	Y
Size (Height)	控件高度	Y	Y	Y
Size (Width)	控件宽度	Y	Y	Y
X	相对于父控件的水平坐标	Y	Y	Y
Y	相对于父控件的垂直坐标	Y	Y	Y
Index(X-Axis)	选项卡视图中选项卡的水平索引	Y	Y	Y
Index(Y-Axis)	选项卡视图中选项卡的垂直索引	Y	Y	Y

备注：

1. 选项卡视图的宽度和高度不能被修改，会默认设置为屏幕的宽度和高度；
2. 选项卡视图的水平和垂直坐标不能被修改，始终为 0；
3. 选项卡视图只能作为屏幕控件的子控件使用；
4. 选项卡视图的子控件只能是选项卡；
5. 选项卡的宽度和高度不能被修改，会默认设置为选项卡视图的宽度和高度；
6. 选项卡的水平和垂直坐标不能被修改，始终为 0。

页面（Page）

具有可滚动内容的容器控件。

表 19: 页面（Page）属性

属性	描述	8762D/8763E	8762G/8772G	TBD
Name	控件名称	Y	Y	Y
Size (Height)	控件高度	Y	Y	Y
Size (Width)	控件宽度	Y	Y	Y
X	相对于父控件的水平坐标	Y	Y	Y
Y	相对于父控件的垂直坐标	Y	Y	Y

备注:

1. 页面只支持垂直滚动;
2. 页面控件的宽度和高度只定义了对滑动手势响应的界面区域，是否允许滚动取决于是否将其添加到超出屏幕范围的子控件中。

窗口（Win）

在窗口的宽度和高度定义的区域内，可以响应各种手势，包括点击、长按、按下、释放和滑动，其属性如下表所示。

表 20: 窗口（Win）属性

属性	描述	8762D/8763E	8762G/8772G	TBD
Name	控件名称	Y	Y	Y
Size (Height)	控件高度	Y	Y	Y
Size (Width)	控件宽度	Y	Y	Y
X	相对于父控件的水平坐标	Y	Y	Y
Y	相对于父控件的垂直坐标	Y	Y	Y
Hidden	表示是否隐藏窗口及其子控件	Y	Y	Y

设计视图/画布

用户可以从工具箱面板中拖放控件到设计视图中，调整控件的布局并设置属性，以设计一个可以在瑞昱 IC 中渲染的图形界面。

图 4: 设计视图

选项卡视图 - 创建/删除/插入选项卡

从工具箱中拖放选项卡视图控件到设计视图中，创建出一个只包含一个主页选项卡（坐标（0,0））的选项卡视图，如下图所示。

图 5: 创建选项卡视图（TabView）

创建选项卡

可以通过点击设计视图周围的按钮来创建新的选项卡。

备注:

1. 如果 idx 为 0，则上下按钮可用；
2. 如果 idy 为 0，则左右按钮可用。

删除选项卡

选择要删除的选项卡，在菜单栏上单击 编辑 ▸ 删除或按键盘上的 `Delete`，然后再次确认是否要删除它。

图 6: 删除选项卡（Tab）确认

插入选项卡

目前，选项卡的插入只支持通过修改现有选项卡的坐标并创建新选项卡来实现。

例如，需要在坐标（1,0）和（2,0）的选项卡之间插入一个选项卡，步骤如下。

1. 将选项卡（2,0）及其右侧的选项卡的 idx 加一，如下图所示；
2. 切换到选项卡（1,0），点击创建新的选项卡（2,0）。

图 7: 选项卡（Tab）插入位置

图 8: 修改选项卡（Tab）坐标点 X 和 Y

图 9: 插入选项卡（Tab）

选项卡视图概览图

请点击查看 选项卡视图概览图。

备注:

1. 在概览图中，高亮显示的选项卡表示当前在设计视图中正在编辑的选项卡；
2. 概览图使用选项卡的坐标进行标注。当在 IC 中进行模拟或渲染时，坐标为（0, 0）的选项卡显示在主页上，用户可以上下左右滑动以显示其他选项卡。

图 10: 选项卡视图（TabView）缩略图

图 11: 选项卡视图（TabView）缩略图

设计视图的缩放

有三种方式可以对设计视图进行缩放。

1. 按住 `Ctrl` 键，并滚动鼠标滚轮；
2. 点击 - 和 + 按钮；
3. 拖动滑动条。

图 12: 设计视图缩放

属性视图

在控件树或设计视图中选择一个控件，可以显示出所有控件的属性值，用户可以根据需要进行修改。

图 13: 属性视图

控件树

控件树用于向用户展示当前布局中控件之间的父子关系和兄弟关系。我们遵循以下约定：

1. 子控件位于父控件的上方，即当父控件和子控件重叠时，子控件会覆盖父控件；
2. 兄弟控件的图层顺序与添加顺序相关，先添加的控件在底部，后添加的控件在顶部。

下图展示了主页选项卡和灯选项卡中的所有子控件，其中主页选项卡只有一个背景图片控件，而灯选项卡包含一个图片控件和多个开关控件。

图 14: 主页（Home）选项卡

图 15: 灯（Lamp）选项卡

控件树支持以下操作：

1. 选择控件：如果在控件树中选择一个控件，设计视图中对应的控件会聚焦，并在属性视图中显示其属性；
2. 修改父子关系：在控件树中选择一个控件（除了选项卡/选项卡视图/屏幕），并将其拖放到目标控件项上，该控件将成为目标控件的子控件；
3. 修改控件图层：在控件树中选择一个控件（除了选项卡/选项卡视图/屏幕），将其拖放到目标控件项的上方或下方，在设计视图中，该控件将在目标控件的上方或下方；
4. 锁定控件：点击锁定按钮，锁定一个或多个控件。
    1. 如果点击屏幕的锁定按钮，所有屏幕的子控件将被锁定，用户将无法在设计视图中拖动或调整控件大小；
    2. 如果点击选项卡的锁定按钮，所有选项卡的子控件将被锁定，用户将无法在设计视图中拖动或调整控件大小。

图 16: 未锁定

图 17: 锁定

2.4.3 资源管理

只有预导入的图像和字体文件可以被 GUI 项目引用。本章重点介绍如何管理图像和字体资源。图像和字体资源管理器位于设计视图正下方，如下图所示。

图 18: 图片资源管理

图 19: 字库资源管理

图像资源管理

点击打开图像管理器，如下图所示。

图 20: 图片管理窗口

添加图像

通过以下步骤可以将图像添加到 GUI 项目中。

1. 点击，创建一个新的图像文件夹并输入文件夹名称。创建的文件夹位于 GUI 项目目录下的 `Resource\image` 文件夹中。

图 21: 创建图片文件夹

2. 选择创建的图像文件夹，然后点击选择图像（支持多选），将它们添加到文件夹中。添加完成后，图像会被复制到 `Resource\image\home` 文件夹中。

图 22: 选择图片文件夹

图 23: 选择图片

图 24: 添加图片

移除图像/图像文件夹

选择要移除的图像或图像文件夹，然后点击。

重命名图像文件夹

选择图像文件夹，双击并输入一个新名称。

预览图像

选择图像文件夹，右侧区域将显示该文件夹中的所有图像。

图 25: 预览图片

刷新

如果用户在本地操作图像资源而不是通过工具进行操作，可以点击刷新。

备注： 不推荐的做法。

字体资源管理

添加第三方字体

如果需要使用第三方字体（`.ttf`），请先点击导入资源，否则将使用本地安装的字体。

图 26: 字库管理

移除第三方字体

选择要移除的字体，然后点击。

2.4.4 菜单栏

文件

起始页

如果要关闭当前项目并打开一个已存在的项目或新建一个项目，可以通过点击 文件 ▸ 起始页 来打开起始页。点击 打开项目 或选择一个 `.rtkprj` 文件并双击打开已存在的项目，或者点击 创建项目 来创建新项目，请参考如何创建项目 和如何打开项目 。

图 27: 开始页面

保存

保存项目的所有 UI 更改，快捷键是 `Ctrl + S` 。

退出保存

当关闭项目时会弹出提示窗口，如下所示。请点击 确定 进行保存，否则更改将会丢失。

图 28: 关闭并保存工程

编辑

复制/粘贴

1. 点击 编辑 ▸ 复制 以复制所选控件，快捷键是 `Ctrl + C` 。
2. 点击 编辑 ▸ 粘贴 以在设计视图中创建所选控件的副本，快捷键是 `Ctrl + V` 。

删除

点击 编辑 ▸ 删除 以删除所选控件，或按下键盘上的 `Delete` 键。

撤销/重做

撤销：撤销对控件所做的更改，快捷键是 `Ctrl + Z` 。重做：重新对控件进行更改，快捷键是 `Ctrl + Y` 。

转换项目

转换项目窗口用于转换当前项目的 IC 类型和屏幕尺寸/分辨率。

图 29: 转换工程

修改项目名称

修改项目名称窗口用于修改当前项目的名称。请在输入框中输入新名称。

图 30: 工程名称

设置

图像转换设置

图像必须转换才能在 IC 上正确显示，因此用户需要设置正确的转换参数。所有可选参数如下图所示。

图 31: 图片转换

参数描述如下。

扫描模式

可用的选项如下表所示。

表 21: 扫描模式可选项

扫描模式	描述
水平	水平扫描
垂直	垂直扫描

颜色空间

图像的颜色空间，可用的选项如下表所示。

表 22: 颜色空间可选项

颜色空间	描述
RGB565	16 位 RGB 模式 位 4:0 表示蓝色；位 10:5 表示绿色；位 15:11 表示红色。
RTKARGB	16 位 ARGB 模式 位 4:0 表示蓝色；位 9:5 表示绿色；位 14:10 表示红色; 位 15 表示透明度。
RTKR-GAB	16 位 RGAB 模式 位 4:0 表示蓝色；位 5 表示透明度；位 10:6 表示绿色；位 15:11 表示红色。
RGB	24 位 RGB 模式 位 7:0 表示蓝色；位 15:8 表示绿色；位 23:16 表示红色。
RGBA	32 位 RGBA 模式 位 7:0 表示蓝色；位 15:8 表示绿色；位 23:16 表示红色; 位 31:24 表示透明度。
BINARY	2-值（0 或 1）图像

压缩

如果勾选 压缩，请根据需要设置压缩参数。可选的压缩模式如下：

1. 行程长度编码（Run-Length Encoding）

一种无损压缩算法。

如果选择 RLE 作为压缩模式，则需要设置 RLE 级别和 RLE 行程长度参数。

图 32: RLE 阶数为 1

图 33: RLE 阶数为 2

表 23: RLE 阶数

RLE 级别	描述
Level1	1 级压缩
Level2	2 级压缩，基于 1 级压缩的次级压缩

表 24: RLE 行程长度

RLE 行程长度	描述
Byte_1	1 个字节，最大为 255
Byte_2	2 个字节，最大为 255

备注： RLE 行程长度：每个行程（Run）在压缩过程中允许的最大重复字符长度。

2. FastLz:

   一种基于字典和滑动窗口的无损压缩算法，用于压缩具有大量重复值的数据。

3. YUV_Sample_Blur:

   一种结合 YUV 采样和模糊处理的有损压缩算法。

   YUV 采样：保留图像的亮度信息，并只对色度信息进行采样。

   模糊处理：在 YUV 采样后丢弃每个字节的低位，以达到数据压缩的目的。

表 25: YUV 采样模式可选项

YUV 采样模式	描述
YUV444	采样 4 个像素数据到 4 个 Y、4 个 U 和 4 个 V，即每个 Y 对应一组 UV 分量，完全保留 YUV 数据。
YUV422	每 4 个像素数据采样为 4 个 Y、2 个 U 和 2 个 V，即每 2 个 Y 对应一组 UV 分量，数据大小为原始数据的 75%。
YUV411	每 4 个像素数据采样为 4 个 Y、1 个 U 和 1 个 V，即每 4 个 Y 对应一组 UV 分量，数据大小为原始数据的 50%。
YUV422	Y - 亮度；V - 色度

表 26: 模糊处理模式

模糊处理模式	描述
Bit0	不丢弃低位的保存方式
Bit1	每个字节丢弃位 0（保留 [位 7: 位 1]）。
Bit2	每个字节丢弃 [位 1: 位 0]（保留 [位 7: 位 2]）。
Bit3	每个字节丢弃 [位 3: 位 0]（保留 [位 7: 位 4]）。

4. YUV_Sample_Blur+FastLz

   该算法结合了 YUV_Sample_Blur 和 FastLz 算法。

字体转换设置

包括位图字体和矢量字体。支持 Realtek 系列 IC 的字体在下表中显示。

备注： 至少需要创建一个字体转换设置，否则无法在属性视图中为文本类型控件选择字体。

表 27: 字体类型

字体	8762D/8763E	8762G/8772G	TBD
Vector（矢量）	N	N	Y
Bitmap（位图）	Y	Y	Y

要使用位图字体，需要设置以下参数。

图 34: 位图字体的转换参数设置

下表列出了每个参数的描述。

表 28: 支持的字体

参数	描述
Font Setting Name	用户定义的字体设置名称，请确保每次创建不同的字体设置名称。
Font Size	字体大小
Bold	是否为粗体
Italic	是否为斜体
Render Mode	渲染模式，用于表示在转换的 `.bin` 文件中像素的位数
Scan Mode	扫描模式，保存.bin 时有两种扫描方式 H: 水平扫描 V: 垂直扫描
Index Metho	索引方式，转换的.bin 文件重新索引区域的索引方法
Code Page	字库代码页，支持多个代码页
Text Type	文本类型，有以下几种。 范围：如果文本的 Unicode 范围可以预先确定，请选择此类型，并在范围文本框中输入范围。支持多个范围，请将每个范围单独设置在一行中。 注意：只有范围内的字符将被转换并保存到.bin 文件中，可以节省存储空间。 随机：如果文本的 Unicode 范围无法预先确定，请选择此类型。 注意：字体的所有字符都将被转换并保存到.bin 文件中。

有关矢量字体的参数，请参考下图。

图 35: 矢量字体参数设置

导出

如果您已经完成了 GUI 项目的设计，并且想将其烧录到 IC 中，请点击 导出，然后工具将执行以下操作：

1. 图像转换；
2. 字体转换；
3. 将 `.xml` 、`.js` 、图像和字体打包到输出的 `.bin` 文件中。

当上述操作完成后，会弹出一个消息框。

图 36: 导出 `.bin`

`.bin` 文件可以烧录进您的 IC 中。

模拟

在 UI 上模拟项目。

备注： 当您第一次模拟项目时，请先点击 导出，然后再点击 模拟。如果您没有修改任何图像或字体设置，则无需再次点击 导出。

下图显示为模拟器运行效果。

图 37: 运行模拟器

2.4.5 快速入门教程

如何创建项目

图 38: 开始页面

双击运行 `RVisualDesigner.exe`，然后按照步骤（1~4）配置项目，点击 创建项目（5）。创建项目后，GUI 设计窗口将弹出。左侧是组件区域，中间是设计区域，右侧是部件属性设置区域。

图 39: （图形化界面）GUI 设计

备注： 新创建的项目文件位于解决方案文件夹下的项目文件夹中。下图展示了一个示例。

图 40: 工程文件夹

当在设计视图上拖放一个控件，并点击 文件 ▸ 保存或按下 `Ctrl + S` 时，将创建一个 `.rtkui` 文件。

图 41: `.rtkui` 文件

如何编写 JavaScript 代码

项目创建完成后，会生成一个 `xxx.js` 文件，打开该文件，在其中编写 JavaScript 代码以实现控件的事件回调函数。

如何打开项目

图 42: 打开工程

有两种打开项目的方式：

1. 点击 打开项目，选择一个 `.rtkprj` 文件进行打开；

图 43: 选择 `.rtkprj` 打开工程

2. 在 **最近项目**区域选择一个 `.rtkprj` 文件进行打开。

如果最近项目中有所需的项目，则会弹出一个消息窗口。

图 44: 消息窗口

如何打开/关闭项目

点击菜单栏上的 文件 ▸ 起始页。

如何导出/打包项目

图 45: 导出

点击菜单栏中的 导出，导出结果会显示在下图中。

图 46: 导出成功

如何进行模拟

图 47: 模拟

点击菜单栏上的 模拟按钮。

2.4.6 GUI 演示项目

`RVisualDesigner-vx.x.x.x.zip` 中包含一个 Demo 项目。

文件夹 454x454 包含分辨率为 454*454 的项目。

文件夹 480x480 包含分辨率为 480*480 的项目。

图 48: 示例

请按以下步骤运行项目：

1. 根据您的 IC 的屏幕尺寸/分辨率打开相应的项目；
2. 点击菜单栏中的 编辑 ▸ 转换项目，检查 IC 类型。有关详细信息，请参考转换项目 。如果当前项目的 IC 类型与您的 IC 不匹配，请选择目标 IC 类型，输入目标分辨率，然后点击 转换。

图 49: 转换工程

3. 点击菜单栏中的 导出，等待导出结果弹出框的出现。

图 50: 导出的 .bin

将导出的 .bin 文件烧录到您的 IC 中。

2.4.7 JavaScript 语法

窗口 (Win)

- 这是一个容器控件。
- 对窗口控件的操作将影响嵌套在容器中的控件。
- 隐藏窗口将隐藏嵌套的控件。
- 当窗口进行图形变换（例如平移和缩放）时，嵌套的控件将进行一致的变换。
- 这个控件可以监控多个手势。

隐藏窗口

- 这个变量 win 被赋值为 win 标签 heat_win 的句柄。
- 变量 hid``被赋值为 ``win 标签的 hidden 属性的句柄。
- 将 hidden 属性的值设置为 hidden 来实现隐藏。

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

监听手势

- win.onPress 函数使 win 控件能够监控手指触摸屏幕的事件。如果手指在窗口区域内触摸屏幕，将执行参数函数。
- win.onRelease 函数使 win 控件能够监控手指离开屏幕的事件。
- 当手指触摸屏幕时，将执行 winNormalOnPressFunc 函数。
- 当手指离开屏幕时，将执行 winNormalOnReleaseFunc 函数。

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

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

切换窗口

- 实现逻辑是点击当前窗口将隐藏当前窗口并显示另一个窗口。
- 点击在 `'cool_win'` 和 `'heat_win'` 窗口之间进行切换。
- `win.onClick` 函数使 `win` 控件能够监控手指点击屏幕的事件。
- `win.removeAttribute` 这个函数用于移除 `win` 标签的一个属性。当移除了 `hidden` 属性时，`win` 标签对应的控件将被显示。
- 在触摸设备上，当用户在 `win` 区域内触摸一个元素并在短时间内抬起手指时，通常会触发一个点击事件。

```
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");
    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) {}
```

(续下页)

(接上页)

```
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"
```

按钮 (Button)

监听按钮按下事件

- 可以用于开发按键按下高亮效果或者需要快速响应的按钮。
- 监听按压手势，当手指在按钮区域内触摸屏幕时，将触发 `iconNromalOnPressFunc` 函数。

```
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) {},
```

文本 (Text)

编辑文本

- 使用 `textbox.write` 函数。

```
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}
```

拖动条 (Seekbar)

显示当前进度

- 拖动进度条，文本将显示当前进度。
- `seekbar.progress` 函数可以读取和写入进度。
- `seekbar.onPressing` 函数将监听手指在屏幕上保持按压的事件。当手指与屏幕接触时，此参数函数将在每一帧中执行。

```
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)
```

一个拖动进度条从 0 增加到 100% 的动画

- 进度条将显示一个从开始到结束不断进度，然后循环回到起点，创建一个永远移动的进度条。
- `seekbar.setAnimate` 这个函数设置进度条的帧动画，传递的参数是帧动画的回调函数和动画持续时间属性。
- 定义一个对象 `curtainAnimateTiming` 来指定动画的时序属性。`duration` 设置动画一个周期的持续时间，以毫秒为单位。`iterations` 是动画应该重复的次数，-1 表示动画应该无限次重复。

```
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)
```

(续下页)

(接上页)

```
}
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
```

开关 (Switch)

监听两个事件

- 开关控件有两个事件，即被打开触发和被关闭触发。
- `sw.onOn` 这个函数用于注册被打开的事件。
- `sw.onOff` 这个函数用于注册被关闭的事件。

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

打开一个 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()
```

- 这个是 writeSync 函数控制 gpio led 在 RTL87X2G 上的实现。
- 首先获取 GPIO 值和方向，然后使用特定的驱动操作 GPIO LED。
- 参考 [https://www.npmjs.com/package/onoff#usage{]}(https://www.npmjs.com/package/onoff#usage)。

```
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
```

图片 (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) {},
```

应用程序 (App)

API

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

进度条 (Progressbar)

API

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

选项卡 (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 语法

元素

- 元素对应控件
- 元素的属性对应控件的属性（0 值可以忽略）
- 文本内容对应控件实例的名称

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

嵌套

元素的嵌套结构与实际控件的嵌套结构一致

```
<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>
```

规格

元素	属性1	属性2	属性3	属性4	属性5	属性6	属性7	属性8	属性9	属性10	属性11	属性12	属性13	属性14	属性15	属性16	属性17
win	x	y	w	h	hidden												
textbox	x	y	w	h	text	font	fontSize	color	mode	inputab							
img	x	y	w	h	scale	scaleY	rotationAngle	blendMod	opacity	file	folde	duration					
seekbar	x	y	w	h	folde	picture	orientation	centralX	centralY	startDegree	endDegree	reverse	blendMod	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	fontColor	fontSize	text	textX	textY	pictureX	pictureY	mode	blendMod	opacity
script	file																
switch	x	y	w	h	picture	highlightPicture	clickPicture	clickHighlightPicture	pictureX	pictureY	blendMod	opacity	mode	duration			
page	x	y	w	h													
screen	w	h															
grid	x	y	rowNumber	colNumber	rowGap	colGap											
gallery	x	y	w	h	folde	main	centerBg	centerPercent	sideS	sidePosPercent	blendMod	opacity					
animateTransform	type	from	to	dur	repeatCour												
motorizedCurtain	x	y	w	h	switc	switc Close	switc Pause										
keyboard	ime																

属性	简介	值
x	相对左坐标	number
y	相对上坐标	number
w	宽度	number
h	高度	number
hidden	隐藏	hidden
text	文本字符串	string
font	字体文件	file path
fontSize	字号	number
color	三原色十六进制色彩	#RRGGBB
mode(textbox)	文本特效	truncate, verticalscroll, horizontalscroll, transition
mode(icon)	按压高亮效果	normal, fade, scale, fadeScale, array
mode(switch)	按压高亮效果	array
inputable	软键盘	boolean
scaleX	水平缩放比例	number
scaleY	竖直缩放比例	number
rotationAngle	旋转角度	number
blendMode	图片混合模式	imgBypassMode, imgFilterBlack, imgSrcOverMode, imgCoverMode
opacity	不透明度从零到二百五十五	number
file	文件路径	string
folder	文件夹路径	string
duration	动画时长（毫秒）	number
picture	图片文件路径	string
orientation(seekbar)	方向	vertical, V, horizontal, H, arc
orientation(curtain)	方向	middle, up, down, left, right
centralX	圆弧中心横坐标	number
centralY	圆弧中心纵坐标	number
startDegree	圆弧起始角	number
endDegree	圆弧结束角	number
transition	变换特效	normal, fade, scale, fadeScale
idx	水平索引	number
idy	竖直索引	number
scope	范围（从零到一）	number
highlightPicture	高亮图片文件路径	string
fontColor	三原色十六进制色彩	#RRGGBB
textX	文字相对横坐标	number
textY	文字相对纵坐标	number
pictureX	图片相对横坐标	number
pictureY	图片相对纵坐标	number
rowNumber	行数	number
colNumber	列数	number
rowGap	行间距	number
colGap	列间距	number
mainBg	主要背景图片文件路径	string
centerBg	中央背景图片文件路径	string
centerPercent	中央区域占比	number
sideScale	侧边图片默认缩放比例	number
sidePosPercent	侧边图片位置占比	number

续下页

表 29 – 接上页

属性	简介	值
type(animateTrans	动画类型	rotate
from	动画起始值	number
to(animateTransfo	动画终点值	number
dur	动画时长	number
repeatCount	动画重复次数	number
switchOpen	电动窗帘开启按钮名称	string
switchClose	电动窗帘关闭按钮名称	string
pauseOpen	电动窗帘暂停按钮名称	string
ime	输入法	null, pinyin
type(onClick)	点击事件触发的行为类型	jump, control
to(onClick)	行为目标	light, multiLevel
id1	主参数	number
id2	次参数	number

示例

窗口

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

图片

```
<img
    x="80"
    y="70"
    w="303"
    h="239"
    opacity="255"
    file="app/box/resource/new_folder/aa2.bin"
    blendMode="imgFilterBlack"
    rotationAngle="0"
    scaleX="1"
    scaleY="1">image3
</img>
```

2.4.9 中间件

RVD 导出了 SaaA 包。固件需要解析和播放它。

包

资源	XML	JavaScript
图片和字体文件等	描述控件的初始嵌套树结构和特定参数	自定义行为，如触发控件手势事件的行为、外围操作、打印日志等

- 包位于文件系统映像的 `root/app` 文件夹中，固件中的启动程序将遍历这些包并为每个包在屏幕上设置一个启动按钮。单击按钮启动相应的包。

启动器

- 启动器的实现在 `realgui\SaaA\frontend_launcher.c` 文件中。
- 它使用网格控件来布局应用程序的按钮。然后它迭代 `app` 文件夹，查找所有表示应用程序的 XML 文件。
- 启动器获取 APP 的标题和图标，并使用按钮控件来显示它们。注册按钮的点击事件是为了启动应用程序。

XML

- APP 包中的 XML 文件描述了控件的初始嵌套树结构和特定参数。
- 使用 `realgui\3rd\ezXML` 将 XML 转换为 C 语言数据格式。详细信息请参考 ezXML SourceForge 。
- XML 解析器的实现在该文件 `realgui\DOM\XML_DOM.c``中。您可以在 ``XML syntax` 页面上阅读语法说明。
- 根据语法规定，函数 `foreach_create` 使用递归策略遍历 XML 的每个标签，并将标签映射到控件，将标签的属性配置给控件。
- XML 遍历完成后，在固件中实际上创建了一个 C-APP，与直接使用 C-APP API 的结果没有区别。
- 然后将执行 XML 中提到的 JavaScript 文件。

JavaScript

- JavaScript 描述了自定义行为，例如触发控件手势事件、外围操作、打印日志等。
- 基于 `realgui\3rd\js` 上的 JerryScript 引擎，支持常见语法。详细信息请参考 JerryScript 。
- JavaScript 解析器的实现在该文件夹 `realgui\SaaA` 中以.js 开头的文件中。您可以在 `JavaScript syntax` 页面上阅读语法说明。
- 使用 `DECLARE_HANDLER` 将函数定义为 JavaScript 函数的 C 语言实现。
- 使用 `REGISTER_METHOD` 和 `REGISTER_METHOD_NAME` 将函数添加到 JavaScript 对象中，以便在脚本中使用它。

- 在 JavaScript 文件中，有一些变量定义、函数定义和函数调用。当应用程序启动时，如上所述，JavaScript 文件将在 XML 解析结束时执行，其中的函数调用将被执行，主要是一些初始化行为和事件监听器的注册。
- 直到事件发生，那些事件的回调函数才会被执行。

示例

进度条 API

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

定义一个进度条对象

实际上，该对象是添加到全局对象中的。使用全局对象的属性不需要显式调用全局对象。

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

向进度条对象添加两个函数

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

定义两个函数

- `progress` 函数用于设置和读取进度条的进度。
- 输入形式参数在数组 `args`` 中，其中第一个是进度数值。如果该参数存在，则表示需要设置进度。使用 ``jerry_get_number_value()` 将 Javascript 参数转换为 C 语言变量。
- 返回值是要获取的进度，使用 `jerry_create_number` 将 C 语言变量转换为 Javascript 变量。需要注意的是，这些 JavaScript 变量在 C 语言中的形式是无符号整数的索引。

```
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);
}
```

- `getElementById` 函数用于获取标签的句柄。更多用法请参考 getElementById on MDN 。

- 输入形式参数是标签的指定名称。使用 `js_value_to_string` 函数将 JS 形式的名称转换为 C 形式的 char 数组，并获取指针句柄，并将值赋给标签。与标准函数定义略有不同，此函数返回新实例化的标签。

```
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();
}
```

灯控制

本页展示了 UI 开关与外设开关之间的对应关系。

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

灯开关数据

数据	值类型	简介
gpio	数字	灯的索引
方向	输出/输入	信号方向
写入的值	数字	0 表示关闭 / 1 表示打开

- 更多信息请参考 onoff npm package usage 。

GPIO 灯开关

- 获取 gpio 索引、方向和写入值；
- 使用 gpio 驱动的 `drv_pin_mode()` 和 `drv_pin_write()` 来操作灯。

MATTER 灯开关

- 获取 gpio 索引、方向和写入值；
- 将数据转换为 MATTER 协议；
- 使用 `matter_send_msg_to_app()` 来操作灯。

MESH 灯开关

- 获取 gpio 索引、方向和写入值；
- 将数据转换为 MESH 协议；
- 使用 `matter_send_msg_to_app()` 来操作灯。

下面的代码示例是基于 RTL87X2G 的 `writeSync` 控制灯的实现。先获取 gpio 值和方向值，然后使用特定的驱动 API 来操作灯。

```
#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

#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]))
```

(续下页)

(接上页)

```
    {
        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;
                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)
```

(续下页)

(接上页)

```
                {
                    //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 3

控件

表 1: 缩略语

词	定义
acc	加速
addr	地址
att	属性
ax	x 轴上的绝对坐标
blit	位块图像传输
buff	缓冲区
cb	回调函数
cbsize	立方体大小
ctor	构造函数
cur_idx	x 方向上的当前索引
cur_idy	y 方向上的当前索引
cx	x 轴上的中心坐标
dc	显示画布
dur	持续时间
dx	触摸板沿 x 轴的偏移
fd	文件描述符
fg	前景
fs	文件系统
hw	硬件
id	索引
img	图像
info	信息
init	初始化
mem	内存
mq	消息队列
nz	平面 Z 方向的法向量
obj	对象
off	偏移

续下页

表 1 – 接上页

词	定义
pic	图片
pos	位置
prev	先前的
rst	结果
src	源
sx	x 方向缩放
tmp	临时的
tx	x 方向的平移

3.1 对象 (Obj)

对象控件实现了屏幕上控件的基本属性。屏幕控件是控件树的根节点。极坐标的原点在 Y 轴的负方向，极坐标的正方向为顺时针方向，屏幕坐标系设置如下所示。

3.1.1 用法

表 2: Gui_Obj 表格

描述	API
构建控件	gui_obj_ctor
创建控件	*gui_obj_create()*
添加事件	gui_obj_add_event_cb
设置事件	gui_obj_enable_event
释放树（从根节点到叶节点，递归地释放控件树）	gui_obj_tree_free
打印树（从根节点到叶节点，递归地打印控件树）	gui_obj_tree_print
获取树中某种类型控件的数量	gui_obj_tree_count_by_type
隐藏/显示	gui_obj_tree_show
能否显示	*gui_obj_show()*
获取根节点	gui_obj_tree_get_root
获取子节点	gui_obj_get_child_handle
判断是否在矩形范围内	*gui_obj_in_rect()*
跳过父对象控件的所有操作（左/右/下/上拖动保持动作）	• 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
跳过子对象控件的所有操作（左/右/下/上拖动保持动作）	• 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
跳过其他对象控件的所有操作（左/右/下/上拖动保持动作）	• gui_obj_skip_other_left_hold • gui_obj_skip_other_right_hold • gui_obj_skip_other_down_hold • gui_obj_skip_other_up_hold
获取显示区域	*gui_obj_get_area()*
矩形范围内的点检测	*gui_obj_point_in_obj_rect()*
CRC 校验	*gui_obj_checksum()*
通过名称在树中获取控件	gui_obj_tree_get_widget_by_name
通过类型在树中获取控件	gui_obj_tree_get_widget_by_type
刷新动画效果	animate_frame_update
设定动画效果	gui_obj_set_animate
以广度优先搜索的方式打印树	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.

返回

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.

返回

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.

参数

- **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.

返回

gui_obj_t*.

void __gui_obj_show__(void *obj, bool enable)

set object show or not.

参数

- **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.

参数

- **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.

参数

- **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.

返回

true.

返回

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.

参数

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.

参数

- **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.

参数

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

返回

true.

返回
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.

参数

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

返回
true.

返回
false.

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

do crc check.

参数

- **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.

返回
uint8_t.

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

print name by bfs order.

参数
object –widget pointer.

返回
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.

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

参数

- **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.

参数

- **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.

参数

- **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.

返回

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.

参数

- **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.

参数

- **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.

参数

- **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.

参数

- **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.

参数

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

返回

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.

参数

- **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.

参数

- **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.

参数

- **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 图像 (Img)

图像控件是用于显示图像的基本控件，支持移动、缩放、旋转等功能。

3.2.1 用法

创建控件

开发者可以使用*gui_img_create_from_mem()* 从内存中创建一个图像控件，或者使用*gui_img_create_from_fs()* 从文件系统中创建一个图像控件。同样，也可以使用*gui_img_create_from_ftl()* 从闪存中创建一个图像控件。如果图像控件的宽度或高度设置为 0，那么控件的大小将根据图像源的大小自动设置。

更新位置

如果想要更新图像控件的位置，开发者可以使用*gui_img_set_location()* 去更新位置。

设定属性

开发者可以通过*gui_img_set_attribute()* 来设置图像控件的属性，替换为新图像并设置新坐标。

获取高度/宽度

如果想要获取图像控件的高度/宽度，开发者可以使用*gui_img_get_height()* 或*gui_img_get_width()*。

刷新大小

开发者可以调用*gui_img_refresh_size()* 来刷新图像控件大小。

混合模式

开发者可以使用*gui_img_set_mode()* 来设定图像控件的混合模式。

移动

通过*gui_img_translate()* 来移动图像控件。开发者可以将图像控件移动到新坐标，而不改变控件属性中的原始坐标。

旋转

开发者可以通过*gui_img_rotation()* 来围绕圆心旋转图像控件。

缩放

开发者可以使用*gui_img_scale()* 调整图像控件的大小以满足需求。

不透明度

图像控件的不透明度值是可调整的，开发者可以调用*gui_img_set_opacity()* 来调整。

动画效果

开发者可以通过*gui_img_set_animate()* 来设定图像控件的动画效果。

质量

开发者可以调用*gui_img_set_quality()* 来设定图像控件的显示质量。

截屏

开发者可以使用*gui_img_tree_convert_to_img()* 来保存全屏截图。保存的图像会是 RGB 格式。

3.2.2 示例

```
#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);

    gui_text_t *text1 = gui_text_create(parent, "text1", 10, 100, 300, 30);
```

(续下页)

(接上页)

```
    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);
}
```

(续下页)

(接上页)

```
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.

参数

_this –the image widget pointer.

返回

uint16_t image’ s width.

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

load the image to read it’ s hight.

参数

_this –the image widget pointer.

返回

uint16_t image’ s height.

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

refresh the image size from src.

参数

_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.

参数

- **_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.

参数

- **_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.

参数

- **_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.

参数

- **_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.

参数

- **_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.

参数

- **_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.

参数

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

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

skew image on Y-axis.

参数

- **_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.

参数

- **_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.

备注: creat an image widget and set attribute.

参数

- **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.

返回

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.

备注: creat an image widget and set attribute.

参数

- **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.

返回

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.

参数

- **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.

返回

gui_img_t*.

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

set animate.

参数

- **_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.

参数

- **_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.

参数

- **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.

参数

img –pointer to the GUI image object.

返回

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.

参数

img –pointer to the GUI image object.

返回

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.

参数

img –pointer to the GUI image object.

返回

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.

参数

img –pointer to the GUI image object.

返回

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.

参数

img –pointer to the GUI image object.

返回

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.

参数

img –pointer to the GUI image object.

返回

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.

参数

img –pointer to the GUI image object.

返回

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*.

参数

- **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.

参数

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

返回

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 文本 (Text)

文本控件是用于显示文本的基本控件，可用于将不同字体、不同颜色和不同大小的文本输出到屏幕上。为了绘制文本，字体文件可以是标准的.ttf 文件或自定义的.bin 文件。

3.3.1 特性

文本控件能支持下面的特性。

- 支持 UTF-8/UTF16/UTF-32
- 支持多语言
- 支持文本排版
- 自动换行和文本滚动
- 抗锯齿
- 支持多种字体
- 支持多种字体大小
- 支持 32 位真彩色
- 支持自定义动画效果
- 支持 EMOJI 表情
- 支持标准 TTF 格式字体 [①]
- 支持自行开发的字体文件

[①]: 只有部分芯片支持此功能

3.3.2 用法

调用对应的函数加载字体文件并显示文本。

初始化字体文件

为了绘制文本，包含字体信息的字体文件需要加载到系统中。

字体文件可以是标准的.ttf 文件或自定义的.bin 文件。字体文件需要提前初始化，且必须为文本控件设置字体类型。

- 初始化自定义 bin 字体文件，需要调用 `gui_font_mem_init(uint8_t *font_bin_addr)`。
- 初始化标准 TTF 文件来绘制文本，需要调用 `gui_font_stb_init(void *font_ttf_addr)`。

所有自定义 bin 字体文件都可以从 RTK FAE 那里获得。

`FONT_BIN`, `FONT_TTF` 这两个文件储存了 flash 中的文件地址。

创建文本控件

开发者可以调用*gui_text_create()* 来创建文本控件。创建后，控件的坐标和文本框的大小已经确定。这些属性也可以随时修改。

备注： 文本控件的大小应大于要显示的字符串，超出范围的文本将被隐藏。

设置文本属性

设置文本

开发者可以调用*gui_text_set()* 来设置文本控件文本、文本类型、颜色、长度和文本字体大小。

备注： 文本长度必须与设置的字符长度相同，文本字体大小必须与加载的字体文件的大小相同。

字体类型

文本控件支持类型设置。开发者可以调用*gui_text_type_set()* 来设置类型。类型为 bin/ttf 文件的地址。

文本内容

开发者可以调用*gui_text_content_set()* 来设置文本控件需要显示的内容。

文本编码

文本控件同时支持 UTF-8 编码、UTF-16 编码和 UTF32 编码输入格式，开发者可以使用*gui_text_encoding_set()* 更改编码方式。

文本转换为图片

使用*gui_text_convert_to_img()* 可以将文本控件中的文本将被转换为图像，存储在内存中，并使用该图像进行呈现。它还支持图像转换，如缩放和旋转。这只适用于位图字体。

备注: 因为需要文本控件的内容和字体大小信息，所以应该在 set text 之后调用它。如果修改了文本的内容、字体大小、位置和其他属性，则需要重用此接口进行转换。

文本输入设置

开发者可以使用函数*gui_text_input_set()* 设置文本的输入。

设置文本点击事件

开发者可以调用函数*gui_text_click()* 添加文本点击事件。

文本模式

文本控件支持七种排版模式，通过*gui_text_mode_set()* 来设置文本排版模式。

排版模式如下：

表 3: 文本模式

类型	行类型	X 方向	Y 方向	控件
LEFT	单行	左对齐	顶部	文本控件 (默认)
CENTER	单行	居中	顶部	文本控件
RIGHT	单行	右对齐	顶部	文本控件
MULTI_LEFT	多行	左对齐	顶部	文本控件
MULTI_CENTER	多行	居中	顶部	文本控件
MULTI_RIGHT	多行	右对齐	顶部	文本控件
MID_LEFT	多行	左对齐	中部	文本控件
MID_CENTER	多行	居中	中部	文本控件
MID_RIGHT	多行	右对齐	中部	文本控件
SCROLL_X	单行	由右向左滚动	顶部	滚动文本控件
SCROLL_Y	多行	左对齐	由下向上滚动	滚动文本控件
SCROLL_Y_REVERSE	多行	右对齐	由上向下滚动	滚动文本控件
VERTICAL_LEFT	多行	左对齐	由上向下	文本控件
VERTICAL_RIGHT	多行	右对齐	由下向上	文本控件

```
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;
```

文本移动

开发者可以调用函数*gui_text_move()* 将文本移动到指定位置，但是 x 和 y 不能大于文本控件的 w 和 h。

设置动画

开发者可以调用函数*gui_text_set_animate()* 来设置动画并在相应的回调函数中实现动画效果。

3.3.3 示例

多文本控件

```
#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);
```

(续下页)

(接上页)

```
    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);
    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);
}
```

动画文本控件

```
#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);
```

(续下页)

(接上页)

```
    }
}

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 .

参数

- **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.

参数

- **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.

备注: The font size must match the font file!

参数

- **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.

返回

void

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

set animate.

参数

- **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.

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

参数

- **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.

参数

- **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.

参数

- **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.

参数

- **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.

参数

- **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.

参数

- **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.

参数

- **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.

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

参数

- **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.

参数

- **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.

备注: The type must match the font size!

参数

- **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.

备注: Need romfs.

备注: Example of a full emoji image file path: “font/emoji/emoji_u1f30d.bin”.

参数

- **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.

备注: utf-8 or unicode.

参数

- **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

备注:

参数

- **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.

参数

- **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.

参数

- **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.

备注: 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.

参数

- **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.

返回

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 模型 (3D Model)

该控件支持加载由 `.obj` 和 `.mtl` 文件组成的 3D 模型，并支持添加动画特效。

3.4.1 GUI 加载 3D 模型

1. 3D 模型组成部分
   - `.obj` 文件：存储 3D 模型的几何数据，包括顶点、法线、纹理坐标、面等。
   - `.mtl` 文件：描述 3D 模型材质的属性，包括颜色、光泽度、透明度和纹理映射等。
   - 图片文件：模型中使用到的贴图。

图 1: 3D 模型组成示例

2. 3D 模型解析并生成 3D 信息描述子
   - 调用脚本处理 `.obj` 文件。

图 2: 脚本处理

   - 生成 3D 信息描述子，该文件包含 obj 解析数据、mtl 解析数据以及纹理贴图。

图 3: 生成二进制数组

3. GUI 加载描述子

   将包含 obj 解析数据、mtl 解析数据和图片数据的 desc 文件放入工程目录下，并在*gui_3d_create()* 中加载。

   示例:

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

3.4.2 3D 控件用法

创建控件

使用*gui_3d_create()* 创建 3D 模型，导入的 `desc_addr` 文件即为脚本中提取的解析数据。

全局形状变换

使用*gui_3d_set_global_shape_transform_cb()* 对 3D 模型进行整体变换，其中 cb 为物体的所有面设置相同的形状变换。该函数中的 world 和 camera 代表了 3D 对象的世界坐标变换和相机视角投影，矩形面还支持设置 light 光照信息。

局部形状变换

使用*gui_3d_set_local_shape_transform_cb()* 对 3D 模型进行局部变换，其中 cb 可以为物体的每个面设置不同的形状变换，face_index 为指定变换的面。该函数中的 world 和 camera 代表了 3D 对象的世界坐标变换和相机视角投影，矩形面还支持设置 light 光照信息。

世界变换

初始化函数为 gui_3d_world_inititalize(gui_3d_matrix_t *world, float x, float y, float z, float rotX, float rotY, float rotZ, float scale)。

- world：指向世界变换矩阵的指针，将 3D 对象从模型坐标系转换到世界坐标系。
- x：沿 X 轴进行平移的距离，用以确定对象在世界坐标系中 X 方向上的位置。
- y：沿 Y 轴进行平移的距离，用以确定对象在世界坐标系中 Y 方向上的位置。
- z：沿 Z 轴进行平移的距离，用以确定对象在世界坐标系中 Z 方向上的位置。
- rotX：绕 X 轴旋转的角度（单位：度）。
- rotY：绕 Y 轴旋转的角度（单位：度）。
- rotZ：绕 Z 轴旋转的角度（单位：度）。
- scale：统一缩放系数，用于在各个方向上等比例地缩放对象。

作用：

1. 世界变换矩阵通常负责将模型坐标系转换到世界坐标系。例如，如果有一个物体位于模型坐标系的原点，通过世界变换，它可以被放置到场景中的任意位置并进行旋转缩放。
2. 对每个面进行独立的世界变换可以实现局部动画或静态展示。
3. 不同的面可以共享同一个世界矩阵,也可以使用 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) 为每个面生成不同的矩阵以实现个性化的局部变换。

相机变换

初始化函数为 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：指向相机结构体的指针，用于初始化相机的属性。
- cameraPosition：相机在世界坐标系中的位置。
- cameraTarget：相机所指向的目标点，即相机视线的焦点。
- near：近裁剪平面距离，定义了相机截取视锥体的近端平面到相机的距离，所有靠近相机而小于这个距离的物体将被裁剪掉。

- far：远裁剪平面距离，定义了视锥体远端平面到相机的距离，所有远离相机而大于这个距离的物体将被裁剪掉。
- fov：视野范围，通常以垂直角度（单位：度）表示，定义了相机的开阔程度，即相机视锥体的张开角度。
- viewPortWidth：视口的宽度，定义渲染目标或窗口的横向尺寸。
- viewPortHeight：视口的高度，定义渲染目标或窗口的纵向尺寸。

作用：

1. 相机变换定义了观察者在场景中的位置和方向，它将世界坐标系转换到摄像机坐标系。
2. 通过操作相机，可以实现不同的视角，例如平移摄像机位置和改变观察方向。

光照信息

初始化函数为 `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：指向光源结构体的指针，用于初始化光源的属性。
- lightPosition：光源在世界坐标系中的位置。
- lightTarget：光源的目标位置，定义光源照射的方向。
- included_angle：光的锥形角度（单位：度），即图中的 α 角，它决定聚光灯的光照范围，即图中聚光灯的外圈范围。
- blend_ratio：光照融合区比例，定义聚光边缘的柔和度，范围为 0~1，决定图中的 β 角，其值通过以下公式计算得出：

$$= (1 - ratio)$$

  融合区域即为下图中的聚光灯内圈到外圈的范围，在内圈内光照强度一致，内圈到外圈光照强度逐渐衰减。

- color：光源的颜色，格式为 RGBA8888。

图 4: 聚光灯效果示例

作用：

1. 光源类型为聚光灯，其属性包含初始位置、光源朝向、锥形角度、融合区比例和光源颜色。
2. 对每个面或对象局部调整光照可以营造不同的视觉风格。

设置动画

gui_obj_create_timer() 函数可以为 3D 对象设置动画属性，其中 `callback` 为动画更新的回调函数。

3.4.3 示例

3D 蝴蝶

该模型全部由矩形面组成，调用*gui_3d_set_local_shape_transform_cb()* 为不同面设置局部变换可以制作动画效果。

```
#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;
```

(续下页)

(接上页)

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

        if (distance > 10.0f)
        {
            // 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);
```

(续下页)

(接上页)

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


    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 棱镜

该模型全部由矩形面组成，调用 gui_3d_light_inititalize() 可以添加光照效果。

```
#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, 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 人脸

该模型由 1454 个三角形面组成。

```
#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)
```

(续下页)

(接上页)

```
{
    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

参数

- **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

返回

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

参数

- **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

参数

- **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.

参数

- **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 视图 (View)

视图控件 (View) 是一种切换更为便利的容器控件，通过事件响应（点击以及四个方向的滑动等）可以实时创建任一视图控件并可选择多种切换效果。切换过程中内存中会存在两个 view，切换完成后会自动清理未显示的 view，可以有效降低内存消耗。

3.5.1 用法

注册视图控件描述子

使用*gui_view_descriptor_register()* 函数传入视图控件描述子地址将其注册在描述子列表中供其它视图控件读取使用，作为创建视图控件的参数，其中 `gui_view_descriptor` 结构体定义如下：

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

(续下页)

(接上页)

```
    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
```

获得视图控件描述子

使用*gui_view_descriptor_get()* 函数通过传入字符串获取对应 name 的视图控件描述子。

创建视图控件

使用*gui_view_create()* 函数可以根据描述子创建一个视图控件。

设置视图切换事件

使用*gui_view_switch_on_event()* 设置视图切换事件，对某一个事件可以重复设置，会使用最新的描述子。具体的事件类型包括 GUI_EVENT_TOUCH_CLICKED 、GUI_EVENT_KB_SHORT_CLICKED 、GUI_EVENT_TOUCH_MOVE_LEFT、GUI_EVENT_TOUCH_MOVE_RIGHT 等。具体的切换风格请参考下列枚举：

```
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;
```

立即切换视图

使用*gui_view_switch_direct()* 立即切换视图，可以配合视图控件中子控件的事件或动画使用，注意切换风格仅限于动画风格，不可设置滑动风格。

获取当前显示的视图控件指针

使用*gui_view_get_current_view()* 函数可以获取当前显示的视图控件指针，可以搭配*gui_view_switch_direct()* 使用，将当前 view 切换。

3.5.2 示例

视图控件

以下是三个单独的 C 文件，每个 C 文件中包含 view 控件的描述子以及 design 函数。

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.

参数

- **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

返回

return the widget object pointer.

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

Register view’s descriptor.

参数

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.

参数

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.

参数

- **_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.

参数

- **_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.

返回

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 4

移植

移植包括平台移植和显示方案的扩展两部分。其中显示方案扩展目前支持字库移植。

4.1 平台移植

移植文件位于 `gui_port` 文件夹中。需要修改六个文件，其文件名与功能如下。

文件名	功能
`gui_port_acc.c`	加速
`gui_port_dc.c`	显示设备
`gui_port_filesystem.c`	文件系统
`gui_port_ftl.c`	闪存转换层
`gui_port_indev.c`	输入设备
`gui_port_os.c`	操作系统

目前已在 FreeRTOS、RT-Thread 和 Windows 上进行了移植，可供参考。

4.1.1 加速

- 参考 `guidef.h` 和 `gui_port_acc.c`。
- 需要根据平台型号，定义加速绘制接口，一般是 “hw_acc_blit“或者 “sw_acc_blit“。
- 结构体定义如下:

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

4.1.2 显示设备

- 参考 `guidef.h` 和 `gui_port_dc.c`。
- 需要定义屏幕的宽度和高度、帧缓冲区地址和模式、分辨率是否缩放等等，并实现刷新函数，结构体定义参考 `guidef.h`。
- 一个典型的 “gui_dispdev“结构体初始化声明如下：

```
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,
};
```

- 在 `DC_SINGLE` 模式下，帧缓冲区的大小为 `screen_width * screen_height * bit_depth / 8`。
- 在 `DC_RAMLESS` 模式下，使用了两个部分帧缓冲区，大小为 “fb_width*fb_height * bit_depth / 8“，此时的 “fb_height“是分段高度。

4.1.3 支持接口类型

以下表格列出了主流芯片支持的与 LCD 相关的接口。如果您想了解更多信息，请点击特定芯片的名称。

芯片	I8080	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

备注： ‘Y’表示驱动程序已包含在库中。‘NA’表示驱动程序尚未包含在库中。

4.1.4 已验证屏幕驱动

以下表格列出了主流芯片支持的与 LCD 相关的驱动 IC。如果您想了解更多信息，请点击特定芯片的名称。

芯片	EK97	ICNA	NT35	NV30	ST77	ST77	ST77	OTM	SH86	SH86	RM69	ST77	NV3041A
RTL8762I	NA	NA	NA	NA	NA	NA	Y	NA	NA	NA	Y	Y	Y
RTL8763I	NA	NA	Y	NA	NA	NA	NA	NA	NA	Y	NA	NA	NA
RTL8772C	Y	Y	Y	Y	Y	Y	Y	NA	NA	NA	NA	NA	NA
RTL8773I	NA	NA	NA	NA	NA	NA	NA	NA	Y	NA	NA	NA	NA

备注： ‘Y’表示驱动程序已包含在库中。‘NA’表示驱动程序尚未包含在库中。

4.1.5 文件系统

- 参考 `guidef.h` 和 `gui_port_filesystem.c`
- 需要定义几个类似 posix 风格的接口操作文件和文件夹。
- 不使用文件系统时可以填入空指针。
- 结构体定义如下：

```
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);
};
```

4.1.6 闪存转换层

- 参考 guidef.h 和 gui_port_ftl.c
- 需要定义闪存转换层的三个接口：read，write，erase。
- 不使用闪存转换层时可以填入空指针。
- 结构体定义如下:

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

4.1.7 输入设备

- 参考 guidef.h 和 gui_port_indev.c
- 输入设备包括触摸板、键盘和滚轮，输入信息的结构体如下：

```
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;
```

- 如果需要某一种输入设备，需要在 “gui_indev“中实现对应的数度获取函数，并填写所需的时间阈值。

4.1.8 触摸芯片

以下表格列出了所有芯片支持的与触摸相关的 IC。如果您想了解更多信息，请点击特定芯片的名称。

芯片	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

备注： ‘Y’表示驱动程序已包含在库中。‘NA’表示驱动程序尚未包含在库中。

4.1.9 操作系统

- 参考 guidef.h 和 gui_port_os.c
- 需要定义线程、定时器、消息队列和内存管理的接口，结构体定义如下：

```
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;
```

休眠管理

为了降低功耗和增加设备的使用时间，支持睡眠（低功耗）模式。

- 参考 gui_app.h

```
typedef struct gui_app gui_app_t;
struct gui_app
{
    gui_obj_t screen;               //!< 控件树的根节点
    const char *xml;                //!< 控件树的设计文件
    uint32_t active_ms;             //!< 屏幕关闭延时
    void *thread_id;                //!< 线程句柄（可选）
    void (* thread_entry)(void *this); //!< 线程入口函数
    void (* ctor)(void *this);      //!< 构造函数
    void (* dtor)(void *this);      //!< 析构函数
    void (* ui_design)(gui_app_t *); //!< UI创建入口函数
    bool lvgl;
    bool arm2d;
    bool close;
    bool next;
    bool close_sync;
};
```

active_ms 是 gui 应用程序的待机时间，可以在不同的应用程序中定义为不同的值。与其他类型的电子设备一样，当屏幕持续显示一个界面的时间超过待机时间时，设备将进入睡眠模式。在睡眠状态下，通过触摸触摸板、按键或发送消息可以唤醒设备。在芯片手册中，这种外设可以关闭的低功耗状态被称为深度低功耗状态（DLPS）。关于 DLPS 的更多信息，可以在 SDK 的相关指导文档中找到。

4.2 字库移植

本章节会解析字库部分代码，并介绍如何使用开发者的字库替换替换 HoneyGUI 的原生字库，或者加入定制化功能。

4.2.1 点阵字库移植

字形加载

文本编码转换

在文件 font_mem.c ，函数*gui_font_get_dot_info()* 中，*process_content_by_charset()* 会解析文本控件的文本内容，并保存为 unicode(UTF-32) 储存到 unicode_buf ，Unicode 数量作为返回值输入 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;
```

(续下页)

(接上页)

```
    return;
}
```

process_content_by_charset() 的具体实现请查阅 draw_font.c 。

备注： 解析过程支持 UTF-8、UTF-16 和 UTF-32。

后续会根据 unicode_buf 中的 Unicode 信息索引字库中的文本数据。

编码转换前后都可以进行小语种的文本编码转换，例如阿拉伯语的字符拼接及其他对 Unicode 进行计算的部分。如果在后面进行转换，需要同步修改 unicode_len 。

备注： unicode_len 的单位是字节，而非字符数量。

字库索引

在文件 font_mem.c ，函数*gui_font_get_dot_info()* 中，解析得到 Unicode 之后，会使用 Unicode 去文本控件指定的字库中索引字形信息。

由于字库工具具有 crop 属性，以及两种索引模式，因此在使用 unicode 在字库文件中寻找文本数据以及点阵数据时，使用了不同的解析代码。

字库解析代码的目的是填充 chr 结构体数组，其结构如下：

```
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;
```

每个成员的含义如下：

- Unicode 是点阵文本的 Unicode，使用 UTF-32LE 格式表达；
- x 是点阵文本边框左上角的 X 坐标，排版时确定，用来确定文本绘制坐标；
- y 是点阵文本边框左上角的 Y 坐标，排版时确定，用来确定文本绘制坐标；
- w 是点阵数据中该字符的数据位宽，由于存在字节对齐以及压缩的特性，因此该值并不总等于字号；
- h 是点阵文本的高度，恒等于字号，用来限制基础绘制区域以及多行排版；
- char_y 是字符的上方空白行数，代表文本点阵图中最上方像素点 Y 坐标与上边框的距离，用来限制绘制区域；
- char_w 是字符的像素宽度，代表文本边框最左侧（起始点）与文本最右侧像素 X 方向坐标差值，绘制时使用该值限制绘制区域，排版时使用该值代表文本宽度；

- `char_h` 是字符的像素高度，代表文本点阵图中最下方像素点 Y 坐标与上边框的距离，用来限制绘制区域；`char_h` 减去 `char_y` 的数值为点阵的实际像素高度；
- `dot_addr` 是该文本对应的点阵数据的起始地址；
- `emoji_img` 是 Emoji 图片对应的控件指针，未使用 Emoji 功能时，该值为空；

图 1: 字形示例

在字库索引阶段，会填充所有 `chr` 的除 `x y` 坐标以外的全部成员，为下一步排版做准备。

备注： 由于不同模式下的数据存储规则有差异，绘制区域也有差异。例如 `char_y` 与 `char_h` 仅在 `crop=1` 并且 `index_method=0` 时才生效。

由于该阶段会使用 Unicode 去查找点阵文本的宽度信息以及点阵数据指针，因此最好在这一步骤之前完成 Unicode 级别的变形文本的融合过程，例如阿拉伯语的拼接，而泰语的字形融合则属于排版阶段的图形融合。

如果使用自己的定制字库进行移植，可以利用定制字库的信息填充至 `chr` 数据结构中，在后续的排版和绘制阶段，使用默认部分。

排版

文本控件支持多种不同的排版模式。

具体的排版功能在文件 `font_mem.c` 的函数*`gui_font_mem_layout()`* 中，每种排版模式具有不同的排版逻辑，但是都依赖于字形信息 `chr` 和文本控件提供的边框信息 `rect` 。

`rect` 结构体数组结构如下：

```
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;
```

`rect` 为控件层传入的控件显示范围，其中 `x1` 和 `x2` 分别代表左边框和右边框的 X 坐标，`y1` 和 `y2` 分别代表上边框和下边框的 Y 坐标，其数值是内部控件计算生成，依赖控件创建时的位置和大小。

通过 `rect` 的四个坐标计算出 `rect_w` 和 `rect_h` 。

四组 `bound` 值是滚动文本控件 `scroll_text` 用来处理显示边界的，文本控件 `text` 暂时没有使用。

开发者可以根据需求，添加新的排版模式。

通过*`gui_text_wordwrap_set()`* 使能了英文单词换行功能（`wordwrap`）后，多行排版会增加英文单词的换行规则，防止英文单词的截断。

字符绘制

点阵字符的绘制代码位于 `font_mem.c` 中的 `rtk_draw_unicode` 中。

可以指定文本控件开启矩阵运算功能以适配文本缩放效果，这部分字符的绘制代码位于 `font_mem_matrix.c` 中的 `rtk_draw_unicode_matrix` 中。

可以指定文本控件开启转图片功能，将文本转化成图片，可以实现复杂特效，这部分字符的绘制代码位于 `font_mem_img.c` 中的 `gui_font_bmp2img_one_char` 中。

字符绘制阶段不涉及任何排版信息，只会读取字形信息，并绘制到屏幕缓存中。

每个字的绘制都会使用控件边框、屏幕的边框以及当前字符的边框三重限制绘制区域。

如果开发者想要使用特殊的字库进行绘制，需要修改点阵数据解析代码，并将像素绘制到屏幕缓存中。

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.

参数

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.

参数

font_bin_addr –font file address

返回

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.

参数

font_bin_addr –font file address

返回

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.

参数

font_bin_addr –font file address

返回
uint8_t

uint8_t **gui_font_mem_destroy**(uint8_t *font_bin_addr)
Destroy this flot type in font list.

参数
font_bin_addr –font file address

返回
uint8_t

void **gui_font_mem_load**(*gui_text_t* *text, *gui_text_rect_t* *rect)
Preprocessing of bitmap fonts using internal engines.

参数
- **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.

参数
- **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.

参数
text –Widget pointer

void **gui_font_mem_obj_destroy**(*gui_text_t* *text)
GUI_FONT_SRC_BMP text widget destroy function.

参数
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.

参数
- **content** –text pointer
- **font_bin_addr** –font file address
- **charset** –text encoding format

返回
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.

参数
- **content** –text pointer
- **font_bin_addr** –font file address

返回
uint32_t

uint8_t **get_fontlib_by_size**(uint8_t font_size)

Get the fontlib name object.

参数
font_size –font size

返回
uint8_t font lib index

uint8_t **get_fontlib_by_name**(uint8_t *font_file)

Get the fontlib name object.

参数
font_file –font file

返回
uint8_t font lib index

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

text layout by mode

参数

- **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

参数
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.

参数

- **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.

返回

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.

参数

- **utf8** –
- **char_num** –

返回

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.

参数

- **unicode_buf** –
- **len** –
- **file_path** –

返回

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 移植

HoneyGUI 是一个轻量级的嵌入式 GUI 系统，专为 Realtek 系列芯片优化设计。本文档将指导您如何在不同的 Realtek 芯片平台上编译 HoneyGUI 库，包括 Armclang 和 Armcc 两种编译器环境的配置和使用方法。

4.3.1 注意事项

- 确保已正确安装 Keil MDK 和 CMake
- 编译前请确保相关依赖库已安装
- 如遇编译错误，请检查芯片型号是否正确指定
- **检查编译器路径设置：**
  - Armcc 编译器默认路径：C:/Keil_v5/ARM/ARMCC/bin
  - Armclang 编译器默认路径：C:/Keil_v5/ARM/ArmCompilerforEmbedded6.22/bin
  - 若安装路径不同，请在 CMake 配置中相应修改编译器路径

图 2: armcc 默认路径

图 3: armclang 默认路径

4.3.2 编译环境要求

- CMake 3.15 或以上版本
- Keil MDK 5 或以上版本
- Windows 操作系统

4.3.3 Armcc 编译

支持芯片:

- RTL8773E (默认)
- RTL8763E
- RTL8762G
- RTL8763D

编译步骤:

1. 在工程路径 armcc 目录下打开 cmd 窗口，生成构建文件，运行命令 `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
```

备注: 若不指定芯片型号，默认为 RTL8773E。

2. 进入 temp 目录编译项目，运行命令 `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. 安装资源，运行命令 `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. **编译生成的资源文件位置：**
    - 头文件：E:/HoneyGUI/lib/armcc/install/include
    - 库文件：E:/HoneyGUI/lib/armcc/install/lib/gui.lib

4.3.4 Armclang 编译

支持芯片：

- RTL8762G (默认)
- RTL8762D
- RTL8773E
- RTL8773G

编译步骤:

1. 在工程路径 armclang 目录下打开 cmd 窗口，生成构建文件，运行命令 `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
```

备注： 若不指定芯片型号，默认为 RTL8762G。

2. 进入 temp 目录编译项目，运行命令 `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. 安装资源，运行命令 `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. **编译生成的资源文件位置：**
    - 头文件：E:/HoneyGUI/lib/armclang/install/include
    - 库文件：E:/HoneyGUI/lib/armclang/install/lib/gui.lib

4.3.5 工程移植示例

本示例以 RTL8773GWP 仪表盘工程为例。

1. 将编译生成的资源文件复制到工程目录：
    - 复制头文件（.h）到工程的资源目录
    - 复制库文件（gui.lib）到工程的资源目录
2. 工程配置：
    - 在 Keil MDK 中添加头文件路径
    - 在工程设置中链接 gui.lib 库

图 4: 链接头文件路径到工程目录

图 5: 链接库文件到工程目录

CHAPTER 5

示例

为了帮助用户熟悉使用该环境，我们提供了一些示例应用程序。这些示例程序会不断增加，您可以从以下配置中选择适合您的配置，配置文件为 “menu_config.h“。

图 1: 配置选择

每个应用程序的入口点如下：

```
GUI_INIT_APP_EXPORT(app_init);
```

5.1 计算器

本示例通过演示如何开发一个简单的 “计算器 APP”，来介绍设计开发 UI APP 的基本方法和流程。示例中的 “计算器 APP” 其功能和传统的计算器功能一致，应用使用 “button” 控件来获取用户输入，用 “text” 控件来显示输入内容和计算结果。请观看以下演示视频来了解完整功能。

5.1.1 源文件

为了帮助学习和熟悉基本的开发流程，开发者可以在路径 `realguiexamplescreen_448_368` 下获取该示例的源文件 `app_calculator.c`。

5.1.2 操作步骤

1. 声明 APP 的结构体

APP 结构体保存了 UI 的所有信息，开发者应该使用 APP 名称和 UI 设计函数完成初始化。

```
#include <gui_app.h>
static void app_calculator_ui_design(gui_app_t *app);

static gui_app_t calculator =
{
    .screen =
    {
        .name = "calculator",
    },
    .ui_design = app_calculator_ui_design,
};

/*
* Public API to get app structure
*/
gui_app_t *get_app_calculator(void)
{
    return &calculator;
}
```

2. 定义 APP :func:ui_design 函数

APP `void ui_design(gui_app_t *app)()` 函数包含了复杂 UI 的所有控件创建及其配置。在本示例 APP 中加入一个“window”控件，并将所有的“button”控件和“text”控件都其作为子控件完成整个 APP UI 设计。

```
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

本示例演示了如何开发一个 RealUI 86Box APP，开发者可以从中学习和理解开发 UI APP 的基本方法和流程。

5.2.1 源文件

- APP package `realguiexamplescreen_480_480rootappbox`
- UI 设计工程 `RVisualDesigner-v1.0.5.0Demo480x480boxbox480x480.rtkprj`

5.2.2 UI 设计

RVisualDesigner

- RealUI 86Box 使用 RVisualDesigner 来完成 UI 设计，首次使用 RVisualDesigner 请参阅 `RVisualDesigner-v1.0.5.0RTKIOT Visual Designer User Guide EN.pdf` 获取详尽的使开发指南。
- 在图示路径中找到并打开示例的 UI 设计工程:
- 依次点击 *Export* 和 *Simulate* 来完成导出和启动模拟，模拟器窗口启动后将显示 86Box APP 图标，鼠标点击后进入对应的 APP。
- 进入 APP，在模拟器窗口中将看到与 RVisualDesigner 设计工程中一致的 UI 内容，因而该设计模式具有“所见即所得”特点。开发者从 *ToolBox* 中拖拽控件到画布中来创建控件到当前页面，添加图片资源到工程后，控件可以配置并链接到自定义的图片。控件间的层级关系将通过 *Widget tree* 来展示。

5.2.3 Javascript

- 控件非默认的效果和逻辑在当前版本中需要开发者通过 JavaScript 来实现，如控件之间的联动，控件默认交互效果包括 switch 控件的点击切换图片、tab 控件的滑动切换等。请参阅 *JavaScript syntax* 了解基于 JavaScript 的 UI 开发方法。

交互

在 JS 文件 `realguiexamplescreen_480_480rootappboxbox.js` 中，实现了 UI 的控制和交互逻辑。

灯光控制 switch

- 为名为“kitchen_switch”的 switch 控件依次注册了打开和关闭 switch 时的回调函数。当 switch 控件“kitchen_switch”被打开时，其回调函数 `led1OnFunc()` 将会被触发调用。
- 灯光的控制在本示例中被抽象为一个 *Gpio* 对象，每盏灯对应一个 *Gpio* 对象，通过 `writeSync()` 函数为其赋值，其行为在底层中被定义，以适应不同的智能家居通信控制协议和控制方式。

```
// 定义一个 Gpio 对象
var LED1 = new Gpio(0, 'out');
function led1OnFunc(params) {
        if (sleep_flag) {
                LED1.writeSync(0)
        }
}

// 为控件 'kitchen_switch' 注册开关动作回调
sw.getElementById('kitchen_switch')
sw.switch_on(led1OnFunc)
sw.switch_off(led1OffFunc)
```

Tab 跳转 switch

1. 为 tabview 控件注册 tab 切换回调，当 tab 滑动切换时，更新当前 tab 索引值并同步 UI 显示状态。
2. 为每个控制跳转的 switch 注册跳转控制回调函数，回调时传入索引值作为参数表征需要跳转的 tab。
3. 在回调函数中通过 `jump()` 函数来跳转，并同步 UI 显示状态。

```
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 趣戴介绍

趣戴科技是一家依托自研 LiteGfx 框架，充分发挥各类芯片性能，为客户提供跨平台、一站式 GUI 解决方案及丰富绚丽特效产品的软件服务企业。通过自研 2.5D 特效框架，模拟实现 3D 技术并融合了粒子系统物理引擎技术，所有 2.5D 特效都内嵌于 LiteGfx Designer，客户可轻松使用并进行个性化开发，为客户打造与众不同的视觉形象。趣戴科技将不断丰富公司在 2.5D 技术上的产品储备，助力客户在激烈的市场竞争中脱颖而出。其坚信优秀的视觉设计是提升企业品牌价值和市场竞争力的关键。

5.3.2 源文件

趣戴组件作为第三方库的形式集成到 HoneyGUI 中，并作为 RealGUI 引擎的一个控件使用，包含核心 lib，控件适配，平台支持三个部分:

相关代码路径：`HoneyGUI\realgui\3rd\litegfx`

```
|-HoneyGUI-Adapt
|       gui_widget_litegfx.c
|       gui_widget_litegfx.h
|       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
```

控件适配层

此目录下的代码是趣戴为了适配 HoneyGUI 提供的 wrapper 层代码。其中 `gui_widget_litegfx.c/h` 为控件，`tab_` 开头的文件是示例代码，直接使用 user 可以参考示例代码实现自己的上层效果。注意，使用此方法本质仍然是通过 RealGUI 调用引擎显示相关效果，此控件支持 FrameBuffer 的分块方式。

平台适配层

平台适配层依据不同的平台重构了适配平台的一些接口，包括调试日志输出，包括内存管理，包括系统时间相关，包括重载一些 C++ 函数（如 new 等）。

核心 lib

提供 Windows GCC 版本的 `liblx_vglite_gcc.a`，提供嵌入式环境版本 `lx_vglite.lib`，请注意编译器版本。

5.4 状态栏

- 这是一种新样式的状态栏。在非下拉状态时，仅在屏幕顶部用小字体显示实时时间。
- 点击顶部可以下拉状态栏，随着下拉，遮罩颜色逐渐变为不透明，时间文字变大。
- 下拉到一定程度后，状态栏完全展开，显示日期和消息通知。

5.4.1 实现

文件

函数 `static void status_bar(void *parent, gui_obj_t *ignore_gesture)` 位于 “realguiexamplescreen_454_454gui_menuapps_in_menu.c“.

设计

- 在此状态栏中，以窗口控件为根节点。使用矩形绘制函数绘制状态栏的白色半透明背景。嵌套了三个文本框，分别表示时间、日期和通知消息。其中，时间文本框使用函数缓存成图像，因为时间显示需要缩放。矩形背景和日期、通知消息的文本初始时被隐藏。触摸屏交互效果在根节点窗口控件的动画回调函数中实现。
- 在窗口的动画回调函数中，首先将时间和日期文本框的内容更新为实时的时间和日期，格式分别为 “07:55” 和 “Tue, Apr 16”。然后，读取触摸板数据，根据当前触屏信息如手势判断状态栏的显示效果，例如是否隐藏背景、是否隐藏日期和通知、更改背景透明度、时间文本框的缩放比例等。
- `` `status_bar` `` 函数有一个参数 `` `ignore_gesture` ``，它接收一个指向控件的指针。此参数用于解决控件上的手势与状态栏之间的冲突。遇到这种冲突时，通过这段代码 `` `if (ignore_gesture) { ignore_gesture->gesture = 1; }` `` 来停用相应控件的手势交互。这里，将 `` `gesture` `` 属性设置为’1’关闭了该控件的手势响应。

5.5 水果忍者

本示例演示了如何开发一个简单的”水果忍者”APP，您可以从中学习和了解开发 UI 应用程序的基本方法和流程。通过切割水果获得分数，直到切到炸弹游戏结束。观看下面的演示视频，了解其全部功能。

5.5.1 环境需求

请参考入门指南 的安装部分内容。

5.5.2 源文件

为了帮助您学习和熟悉开发，您可以在 `realgui\example\screen_454_454` 路径下找到您可能需要的所有源文件。本演示的源文件是 `app_fruit_ninja_box2d.cpp`，您可以在上述路径中找到它，了解更多详情。

5.5.3 配置

```
#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 调用步骤

步骤 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);
}
```

步骤 2: Call function

```
extern void app_fruit_ninja_design(gui_obj_t *obj);
app_fruit_ninja_design(GUI_APP_ROOT_SCREEN);
```

5.5.5 设计思路

- 在该 app 中，使用到了 box2d 创建固体模拟在重力环境中的物体运动，在初始化时给定一个 x 轴与 y 轴的初速度等参数。

```
/* 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);
```

- 因为相互碰撞不利于游戏的游玩，为了减小物体间相互碰撞的影响，将固体的半径设置为一个较小的值。

```
/* Creat body shape and attach the shape to the Body */
b2CircleShape circleShape;
circleShape.m_radius = 0.2; // Small radius reducing the impact of␣
↪collisions
```

- 在回调函数中利用固体的中心点映射更新水果（及炸弹）的位置与旋转角度并用图片组件显示。水果位置在显示界面外会对固体的位置与初速度进行复位。

```
/* 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);
}
```

- 切割水果使用了 touch_info 结构体，检测到触控点释放说明完成了一次切割（得到触屏初始点与 x 轴与 y 轴的位移），对切割内容进行判断。

```
/* Cutting judgment */
GUI_TOUCHPAD_IMPORT_AS_TP // Get touchpoint
if (tp->released)
{
   bool bomb_flag = cutting_judgment(win, img_strawberry, img_banana,␣
↪img_peach, img_watermelon,
                                     img_bomb, tp, img_cut_arry, fruit_
```

(续下页)

(接上页)

```
↪cut_flag);
}
```

- 若切割线与图片矩形有两个交点，则说明切割有效。

```
line_has_two_intersections_with_rectangle(img_coordinate, img_w, img_h,␣
↪tp_start, tp_end,
                                                                img_rotate_
↪angle);
```

- 注意在计算交点时图片的旋转后端点信息需要将旋转角度带入计算才会与显示一致，如此可以提高切割判断准确度。

```
/* 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
```

- 将水果图片更新为切割后的两张图（对应两个 gui_img_t 指针），并计分，一次切割可以切到多个不同物体。

```
/* 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);
}
```

- 注意可以使用 flag 标记水果的切割状况，防止计分错误以及方便更新切割后的图片位置。
- 当切割后的水果移动到显示界面之外会对固体的位置与初速度进行复位，并将切割效果复原。

```
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 音乐播放器

- UI 设计: Figma - 音乐移动应用 UI
- I 直观的三层设计：在三个不同的界面之间轻松导航。中央界面展示当前曲目的专辑封面以及基本的播放控件。
- 滑动导航：只需简单滑动，即可切换到顶部界面访问您的歌曲列表。
- 歌词显示：向下滑动可以显示全屏歌词界面，同步显示音乐歌词。
- 流畅动画：享受界面间美妙流畅的过渡，以独特的缩放动画让专辑封面生动起来，让您在各种查看选项间切换自如。

5.6.1 实现

代码

函数 `void app_music_ui_design(gui_obj_t *obj)` 位于文件 `realgui/example/screen_454_454/gui_menu/app_music.cpp`。

控件树设计

图 2: 控件树设计

5.7 定时器

- 此应用程序具有两个界面：定时器和秒表，可以通过点击底部的两个按钮轻松切换。
- 定时器界面：点击即可开始计时，屏幕上会显示秒数递增。
- 秒表界面：使用三个可调滚轴选择您的开始时间，包括小时、分钟和秒。开始后，会有一个动画效果，选择的时间居中并开始倒计时。

5.7.1 实现

代码

函数 `app_clock_ui_design` 位于文件 `realgui/example/screen_454_454/gui_menu/app_clock.c`。

控件树设计

图 3: 定时器控件树设计

5.8 表盘市场

- 表盘 UI 设计: Figma - 表盘 UI 界面
- 该应用程序让您可以轻松浏览和安装新表盘以匹配您的风格。
- 轻松浏览：通过清晰的两列布局浏览表盘预览。上下滚动以查看选项。
- 预览和选择：点击任何表盘预览以查看淡出效果，然后选择它将您的手表切换到新表盘。
- 快速访问：长按当前表盘即可快速打开表盘市场应用，探索新的设计。
- 简单安装：要安装新表盘，只需将表盘包复制到智能手表的指定文件夹中。

5.8.1 实现

代码

函数 `GUI_APP_ENTRY(APP_WATCHFACE_MARKET)` 位于文件 `realgui/example/screen_454_454/gui_menu/watchface_market.c` 中。

控件树设计

图 4: 控件树设计

CHAPTER 6

工具

在 UI 设计中，需要利用图片转换工具或字体转换工具将图片或字体转换成二进制文件，然后使用打包工具将所有 UI 资源文件进行打包，最后使用烧录工具进行烧录。本章将介绍这四种工具的用法。

6.1 图像转换工具

6.1.1 图像格式转换

将各种格式的图片转换为 RGB 原始图片。

- 打开转换器。图片转换工具下载链接请参考该章节：工具。
- 操作步骤及详细说明如下：

1. 打开图片文件夹。
2. 打开设置。
3. 选择输出文件夹。
4. 勾选颜色信息头。
5. 选择要进行设置的图片格式。
6. 设置转换参数。
7. 转换。

配置

- Color head : BeeGUI 渲染图像需要用到的头部信息。
- Big-endian : 输入图像是否采用大端模式。
- Compress : 启用图像压缩。
- MixAlphaChannel Flag : 在将 RGBA 转换为 RGB 或 RGB565 时，是否将 alpha 通道混合到 RGB 中。
- Scan Mode : 选择扫描方向是水平还是垂直，BeeGUI 仅需要水平方向。
- Color Space : 选择颜色空间（RGB565、RGBA、BINARY…等）。

颜色空间

- RGB565: 色彩丰富但渲染成本和存储需求低，每个像素占 2 字节存储。

Red	Green	Blue
5bit	6bit	5bit

- ARGB8565: 24 位的 ARGB 格式。

Opacity	Red	Green	Blue
8bit	5bit	6bit	5bit

- RTKRGAB: 16 位的 RGAB 格式。

Red	Green	Opacity	Blue
5bit	5bit	1bit	5bit

- RGB : 24 位 RGB 格式，每个像素占 3 个字节存储。

Red	Green	Blue
8bit	8bit	8bit

- ARGB : 带有透明度的真彩色，通过透明度效果提升显示质量。每个像素占 4 字节存储。

Opacity	Red	Green	Blue
8bit	8bit	8bit	8bit

- BINARY : 使用一个位来表示一个像素。
- RTKARGB8565 : 24 位 ARGB8565 格式。

Opacity	Opacity	…	Red	Green	Blue	Red	Green	Blue	…
8bit	8bit	…	5bit	6bit	5bit	5bit	6bit	5bit	…

6.1.2 输出文件

转换后，将生成以下文件。

通过使用图像转换工具（ImageConvert Tool），开发者可以将三个 JPG 文件（a、b 和 c）转换为 二进制文件（a、b 和 c）。

接下来，开发者需要将这些 二进制文件放置在打包目录的 根文件夹（root）中。关于打包资源的过程，请参考打包工具 章节。

6.2 字体转换工具

字体转换工具功能：从标准内部码表（代码页文件）、自定义 Unicode 码表（或补充码表.txt 文件，自定义.cst 文件）中获取所有待转换字符对应的 Unicode 码。根据 Unicode 编码从字体文件（如.ttf）中找到相应字符的矢量字体数据。将其转换为位图，并输出为.bin 文件。

6.2.1 生成字体 bin 文件

如何生成对应的文件，请参考以下步骤:

1. 将字体库文件复制到该目录 \Font Convert Tool\font。

2. 可以参考 \Font Convert Tool\doc 目录下的文档，了解每个参数的具体含义，通过编辑 FontConfig.json 配置字体参数。

3. 打开 setting.ini，修改可选配置项。

4. 双击 fontDirctionary.exe 生成 font.bin。

6.2.2 FontConfig.json 参数说明

表 1: FontConfig.json 参数说明

字段名称	字段含义
cst 路径 (cstPaths)	二进制 Unicode 代码表 cst 文件路径。可以设置多个。
自定义值 (customerVals)	用户定义的连续 Unicode 字符。可以设置多组。
起始值 (firstVal)	自定义连续 Unicode 字符的起始值。
范围 (range)	自定义的连续 Unicode 字符数量。
映射路径 (mappingPaths)	用户定义的 Unicode 字符集文件路径。可以设置多组。
字体设置 (fontSet)	用于指定要转换的字体相关设置。
加粗 (bold)	指定转换后的字符是否加粗。
斜体 (italic)	指定转换后的字符是否倾斜。
扫描模式 (scanMode)	指定转换后的字符数据如何保存。如果值为 “H”，字体按行保存；如果值为 “V”，字体按列保存。
字号 (fontSize)	指定转换后的字符大小。
字体 (font)	指定用于转换的字体文件。
渲染模式 (renderMode)	指定在转换后的字符位图中表示一个像素使用的位数。支持 1/2/4/8。
索引方法 (indexMethod)	指定转换后输出 bin 文件的重索引区域的索引模式，并用 0 填充地址索引；偏移索引用 1。 当字符数量超过 100 时，建议选择索引模式 0。
裁剪 (crop)	压缩字体文件大小。建议始终开启。目前仅支持 IndexMethod=0 时的裁剪.

6.2.3 Setting.ini 参数说明

表 2: Setting.ini 参数说明

gamma	1	gamma 值是用于描述输入像素值和输出亮度之间非线性关系的参数。该值越大，文本亮度越高。
rotate	0	字体旋转角度。0：不旋转。1：顺时针旋转 90°。2：逆时针旋转 90°。

6.3 打包工具

6.3.1 RTL87x2G and RTL8762D

RTL87x2G 是一系列 IC 类型的缩写。

RTL87x2G 和 RTL8762D 的打包流程一样，以下以 RTL8762G 为例进行说明。

在开始之前，请在 SDK 目录下的（`\subsys\gui\realgui\example\screen_800_480\root_image_800_480`）选择一个合适的示例，或者基于示例创建一个新的打包目录。然后，将 bat 和 py 脚本复制到该目录中，确保 root 文件夹以及 bat 和 py 脚本都处于该目录下。

用户数据生成流程如下：

1. 将所有生成的 `bin` 文件复制到文件夹 `\src\app\dashboard\application\root_image\root` 中。

2. 在 \src\app\dashboard\application\root_image 目录下，双击 mkromfs_0xa00000.bat 脚本文件以执行它，并生成 root 文件夹的镜像。执行后，该目录下将出现一个新的 .bin 文件和 .h 文件。
3. 其中，.h 文件包含了文件系统中每个文件的地址偏移量，可以在不使用文件系统的情况下直接访问。在开发 GUI 代码之前，请将包含 .h 的文件夹添加到包含目录中。
4. 其中，.bin 文件是根文件夹的镜像文件，使用 MPTool 工具将 root_xx.bin 文件烧录到 Flash 存储中。

6.3.2 RTL8763E and RTL8773DO

RTL8763E 是一个 IC 系列的名称，包含 RTL8763EWE-VP 和 RTL8763EW-VC。RTL8763E 和 RTL877DO 打包流程类似。

在开始之前，请在 SDK 目录下的（\tool\Gadgets\gui_package_tool ）选择对应 IC 的目录，8763EW 请选择 8763E 目录，8773DO 请选择 87x3D 目录。

生成用户数据过程如下：

1. 将所有生成的 bin 文件复制到文件夹 \tool\Gadgets\gui_package_tool\8763E\root 中。
2. 双击 \tool\Gadgets\gui_package_tool\8763E 目录中的 gen_root_image.bat 文件以执行脚本并生成 root 文件夹的镜像。一个新的 .bin 文件和 .h 文件将出现在目录中。
3. 其中，.h 文件包含了文件系统中每个文件的地址偏移量，可以在不使用文件系统的情况下直接访问。
4. 其中，.bin 文件是根文件夹的镜像文件，使用 MPTool 工具将 root_xx.bin 文件烧录到 Flash 存储中。

6.3.3 RTL8773E

RTL8773E 是一个 IC 系列的名称，包含 RTL8773EWE 和 RTL8773EWE-VP 等，用户数据打包处理过程如下：

生成用户数据

1. 将生成的图像 bin 文件复制到 \src\app\watch\gui_application\root_image\root\8773e_watch 目录，并将生成的字体 bin 文件复制到 \src\app\watch\gui_application\root_image\root\font 目录。
2. 修改构建地址（build address）：开发者需要通过修改 mkromfs_0x4400000.bat 文件（python_bin_mkromfs_0x4400000.py --binary --addr 0x238b400 root root(0x4400000).bin）。这里的 --addr 参数对应于闪存映射中的 userdata 地址加上 0x400（图像头部大小）。
3. 在 \src\app\watch\gui_application\root_image 目录中双击 mkromfs_0x4400000.bat 文件来执行脚本，并生成 root 文件夹的镜像。一个新的 bin 文件 root(0x4400000).bin 和头文件 ui_resource.h 将出现在目录中。
4. 在这两个文件中，.bin 文件是包含 root 目录内容的镜像文件，而 .h 文件则包含了文件系统中每个文件的地址偏移量，这些偏移量允许开发者在不使用文件系统的情况下直接访问这些文件。

备注： 生成的 ui_resource.h 文件需要手动添加以下代码。

```
#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
```

添加头部信息

使用 MPPG Tool 给用户数据文件添加头部信息，添加过程如下：

1. 选择菜单 *Tool‣Prepend header for user data*。
2. 添加 `flash_map.ini` 路径。
3. 添加用户数据文件（`root_xx.bin`）路径。
4. 生成可烧录的用户数据文件。

备注： 最大 size 必须大于实际 size，否则，需要更改 flash_map 中的用户数据大小。

6.4 烧录工具

MP Tool 支持调试模式和批量生产模式，集成了打包和 flash map 生成功能。

- 调试模式：为开发人员提供一个平台，用于调试和功能开发。
- 批量生产模式：提供一系列功能，包括能够同时支持多达 8 个设备 flash 的下载和修改设备的蓝牙地址。

6.4.1 下载到 EVB 中

在 MP Tool 启动界面选择芯片类型和语言，以 RTL8762G 为例。

图 1: MP Tool 启动界面

- 加载烧录所需的文件，包含 flash map、System Config File、APP Image 等。
- 选择 User Data。

图 2: MP Tool 主界面

- 将生成的图像文件下载到指定的地址（例如文件系统挂载地址），8762G 的地址为 0x04400000。

图 3: User Data 加载界面

- 文件准备完成后，先检测 UART 端口，正常则显示 *Ready*。然后打开 UART 端口，显示 *OK* 后即可点击 *Download* 烧录。

图 4: 进入烧录模式界面

CHAPTER 7

设计说明

本章节介绍了 RealUI 系统的工作流程，包括从输入数据到显示在 LCD 上。

7.1 RealUI 系统概述

RealUI 系统是基于 HoneyGUI 的高效嵌入式显示项目解决方案。

7.1.1 RealUI 工作流程

RealUI 系统的工作流程主要分为四个步骤：

系统

系统初始化主要包括系统时钟的初始化、外设的初始化以及项目的其他模块的初始化，例如*PSRAM*、*LCD*、*TP* 和蓝牙。

GUI 服务器

首先，预先填充的 GUI 端口部分会被初始化，包括操作系统、显示器、输入设备和文件系统。然后创建 GUI 服务器线程，并且 GUI 服务器会在 GUI 线程中持续运行。

GUI 应用程序

GUI 应用程序是由多个控件组成的一系列显示界面。为了运行 GUI 应用程序，需要先启动它。

GUI 服务器任务

GUI 服务器是 GUI 任务的运行函数，它的具体运行过程分为六个部分：

1. GUI 应用程序存在性检查：首先，需要获取当前正在运行的 GUI 应用程序。当 GUI 检测到有正在运行的 GUI 应用程序时，会进入下一步；
2. 获取*LCD* 数据：获取屏幕的实时信息；
3. 获取触摸板数据：获取触摸板的实时信息，并运行触摸算法；
4. 获取*KB* 数据：获取键盘的实时信息，并运行键盘算法；
5. 对象绘制：在应用程序中绘制控件，包括功能操作和图像处理；
6. 更新*FB* ： 将绘制的结果传递到屏幕上显示。

更详细的 GUI 应用程序操作可以在在线文档中找到。

7.2 输入子系统

UI 系统可以接收设备中其他外设的输入，典型的输入设备有触摸板和按钮。

本章节详细介绍了如何在 UI 系统中使用输入设备，并详细描述了输入信息的处理过程。

7.2.1 触摸板

触摸板是最常用的输入设备之一，大部分情况下，触摸板会集成在显示面板中。触摸信息的工作流程如下图所示。

图 1: 触摸板工作流程

触摸板硬件和驱动程序

尽管不同的触摸板芯片具有不同的消息数据结构，但消息始终包含触摸状态和触摸点的坐标。为了传输坐标信息，需要一个数据总线，而*I2C* 是触摸芯片和微处理器之间最常用的数据总线。

此外，根据规格要求，不同的触摸芯片需要使用不同的驱动程序，需要进行移植。

获取触摸板数据

在 port_touchpad_get_data 函数中，触摸信息将在 drv_touch_read 中获取，经过简单处理后，作为原始数据供触摸算法处理程序使用。

```
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)
    {
        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;
}
```

原始数据的数据结构是 gui_touch_port_data_t。

触摸板算法处理器

触摸板算法处理的代码实现在 tp_algo_process 函数中。通过判断 X 轴和 Y 轴坐标数据的变化以及触摸时间，进行手势识别。经过算法处理后得到的输入类型如下：

```
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,
```

(续下页)

(接上页)

```
    TOUCH_INVALIDE              = 0x1FF,

    KB_INIT                     = 0x200,
    KB_SHORT                    = 0x201,
    KB_LONG                     = 0x202,
    KB_INVALIDE                 = 0x2FF,

    WHEEL_INIT                  = 0x300,
    WHEEL_ING,
    WHEEL_FINISHED,
    WHEEL_INVALIDE              = 0x3FF,
} T_GUI_INPUT_TYPE;
```

算法处理器将填充 `touch_info_t` 结构体，该结构体对所有控件可用。

控件响应

一些控件可以对触摸板信息做出响应，例如窗口控件、按钮控件、选项卡控件、窗帘控件和进度条控件。其中，窗口和按钮主要响应点击事件，选项卡、窗帘和进度条主要响应滑动事件。此外，选项卡、窗帘和进度条的显示也取决于 `touch_info_t` 结构体中的实时触摸坐标。

大多数控件在相应的准备函数中处理触摸信息，比如 `win_prepare`。使用 `tp_get_info` 函数获取触摸信息。

在应用程序层面，可以根据不同类型的事件绑定不同的回调函数，示例如下：

```
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);
```

(续下页)

(接上页)

```
    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);
}
```

在这个例子中，首先创建了一个名为 clock 的窗口，在点击时执行 `show_clock` 函数，在长按时执行 `enter_homelist` 函数。

7.2.2 键盘

键盘信息的工作流程如下图所示：

图 2: 键盘工作流程

硬件和驱动程序

键盘的硬件设计和驱动程序比较简单，本章将通过一个单独的*GPIO* 来说明。有关如何使用 GPIO，请参考 SDK 中的说明。您可以使用 `rtl87x2g_gpio.c` 中的通用*API* 或 `drv_gpio.c` 中的封装 API 来完成相同的操作。

获取键盘数据

在 `port_kb_get_data` 函数中，可以获取到键盘信息。用户根据自己的功能需求填写 `port_kb_get_data`，并将结构体填充为键盘输入信息。

键盘算法处理器

键盘算法的代码实现在 `kb_algo_process` 函数中。通过按压时间的长短来确定输入的类型是短按还是长按。算法处理器将填充 `kb_info_t` 结构体，该结构体对所有控件都可用。

响应

对键盘的响应有两种方式，一种是在控件（如窗口）中响应经过处理的按键信息，另一种是在接收到按键时直接响应按下动作。

第一种方式示例如下所示。

```
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);
```

(续下页)

(接上页)

```
    }
......
}
```

对于第二种方式，请参考 GPIO 用户手册。

7.3 显示子系统

显示系统的工作流程非常复杂，不同的 UI 框架和控件有不同的处理过程。

7.3.1 显示工作流程

图片作为最常用的 UI 输入源，这里以图片作为示例，说明从原始图片到屏幕的完整工作流程，如下图所示。

由于不同类型的*IC* 具有不同的硬件配置，这里选择 RTL8772G 芯片平台，并使用 RealUI 作为 UI 系统来解释图片的显示工作流程。

图 3: 图像显示工作流程

Flash 文件系统

原始图片被转换成特定格式的文件，然后下载到 Flash 中。Flash 配置了一个伪文件系统，提供图像索引信息给控件层。完成文件系统的简单迁移后，就可以使用标准文件系统。

请阅读图像转换工具 章节以获取有关图像转换的更多信息。

UI 控件

图像控件是用于显示图像的最基本的 UI 控件。UI 系统中还有许多控件是基于图像控件绘制的特殊图像。

在这里，图像控件加载图像数据并读取图像信息，结合 UI 设计和控件层的行为，为加速层提供图像渲染需求。例如图像移动、图像缩小和放大、图像旋转等。

此外，某些硬件支持功能强大的*GPU*，即图形处理单元，可以绘制具有复杂变换效果的控件，例如立方体控件、色环控件等。

加速层

加速层的功能是加速 UI 图像绘制过程，分为硬件加速和软件加速。通常情况下，硬件加速明显优于软件加速，但使用哪种取决于 UI 系统部署的硬件环境。此外，不同的硬件加速器，也称为图形处理单元（GPU），也具有不同的功能。加速器接收 UI 控件分配的绘制任务，并将完成的图像传输到显示缓冲区。

缓冲区

在大多数嵌入式系统中由于*RAM* 有限，RealUI 采用分块渲染机制，需要使用显示缓冲区。显示缓冲区存储了加速器的图像绘制结果和其他非加速控件的绘制结果，数据通过*DMA* 传输到帧缓冲区。

当可用 RAM 可容纳一个完整的帧时，可以使用单帧绘制模式，此时使用完整的帧缓冲区而不是显示缓冲区。

配置显示控制器后，将数据从帧缓冲区传输到屏幕，此时屏幕将显示 UI 界面。

7.4 软件加速

7.4.1 总体流程图

该流程图描绘了软件加速处理图像资源的流程。在处理图像时，根据图像的压缩状态和类型选择不同的处理方法：

- **覆盖 (cover)**: 将原始图像数据直接写入到帧缓冲区中的相应位置。不进行任何处理，只是直接覆盖。
- **旁路 (bypass)**: 将原始图像数据直接写入到帧缓冲区中的相应位置。旁路模式无法处理图像的透明度。它对整个图像应用全局不透明度值，从而影响整体透明度。在创建透明效果时，旁路模式相比源覆盖模式更占用空间。
- **滤黑 (Filter black)**: 滤波技术有效地从原始图像数据中筛选出像素值为零的像素数据，这意味着黑色像素不会被写入到帧缓冲区中。这种机制能够加快刷新速度。除黑色以外的任何颜色像素都会经过标准处理方法，并被记录到帧缓冲区中。
- **源覆盖 (Source_over)**: 一种混合方法，将图像的颜色数据和帧缓冲区像素的颜色数据结合起来，基于不透明度值 Sa 计算最终的颜色，并将其写入到帧缓冲区的相应位置。计算公式为 ((255 - Sa) * D + Sa * S) / 255)，其中 Sa 是原始图像的不透明度值，D 是帧缓冲区的像素数据，S 是源图像的像素数据。

图 4: 软件加速

- img_type 可以从图像的头部 head 中获得，图像头部的结构如下：

```
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;
```

- img_type 的枚举值如下。如果值为 IMDC_COMPRESS，则表示图像已压缩，并进入 rle 处理流程；否则，进入 no rle 处理流程。

```
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,␣
```

(续下页)

(接上页)

```
↪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;
```

- 根据不同的混合模式 blend_mode 执行相应的 blit 过程。

```
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;
```

- 当图像被压缩时，需要从压缩数据的地址中获取压缩头。该头部中的 algorithm_type 参数包含了实际的图像类型。压缩图像的类型在 imdc_src_type 结构体中描述，包括三种类型：IMDC_SRC_RGB565、IMDC_SRC_RGB888 和 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 无 RLE 覆盖模式概述

以下流程描述了 No RLE 压缩图像的 Cover mode 处理过程。根据图像矩阵和显示设备的像素字节数选择处理方法，并将其写入帧缓冲区。

图 5: Cover Mode Path

- 如果矩阵是单位矩阵，则执行没有矩阵操作的 blit 过程；否则，执行具有矩阵操作的 blit 过程。
- dc_bytes_per_pixel 表示显示设备的像素字节数，计算方式为 dc->bit_depth » 3，其中 bit_depth 为显示设备的位深度。以位深度为 24 的显示设备为例，其像素字节数为 3。

无 RLE 覆盖模式（不带矩阵变换）

下面的流程图描述了将 Uncompressed images 写入帧缓冲区的 Cover mode 处理过程，以 RGB565 为目标设备图像类型为例。

图 6: Cover_blit_2_rgb565

无 RLE 覆盖模式（带矩阵变换）

下面的流程图描述了使用 Cover mode with matrix operations 将 Uncompressed images 写入帧缓冲区的过程，以 RGB565 为目标设备图像类型为例。

图 7: Cover_matrix_blit_2_rgb565

7.4.3 无 RLE 旁路模式概述

以下流程描述了 No RLE 压缩图像的 Bypass mode 处理过程。根据图像矩阵和显示设备的像素字节数选择处理方法，并将其写入帧缓冲区。

图 8: Bypass_mode_path

- 如果矩阵是单位矩阵，则执行没有矩阵操作的 blit 过程；否则，执行具有矩阵操作的 blit 过程。
- dc_bytes_per_pixel 表示显示设备的像素字节数，计算方式为 dc->bit_depth » 3，其中 bit_depth 为显示设备的位深度。以位深度为 24 的显示设备为例，其像素字节数为 3。

无 RLE 旁路模式（不带矩阵变换）

下面的流程图描述了将 Uncompressed images 写入帧缓冲区的 Bypass mode 处理过程，以 RGB565 为目标设备图像类型为例。

图 9: Bypass_blit_2_rgb565

1. 根据 img_type 执行不同的处理步骤。
2. 基于 opacity_value 执行相应的操作将图像像素写入帧缓冲区。

- 如果 opacity_value 为 0，表示图像不显示，直接跳出处理流程。
- 如果 opacity_value 为 255，将源图像像素转换为 RGB565 格式，并写入帧缓冲区。
- 如果 opacity_value 介于 0 和 255 之间，执行 Alpha 混合操作将源图像像素与对应的帧缓冲区像素进行混合。混合公式为 ((255 - Sa) * D + Sa * S) / 255)，将混合结果写入帧缓冲区。

无 RLE 旁路模式（带矩阵变换）

下面的流程图描述了使用 Blend mode with matrix operations 将 Uncompressed images 写入帧缓冲区的过程，以 RGB565 为目标设备图像类型为例。

图 10: Bypass_matrix_blit_2_rgb565

1. 根据 img_type 执行不同的处理步骤。
2. 执行矩阵计算，将目标区域的写入点映射到图像像素，并获取图像像素的像素值。
3. 基于 opacity_value 执行相应的操作将图像像素写入帧缓冲区。
   - 如果 opacity_value 为 0，表示图像不显示，直接跳出处理流程。
   - 如果 opacity_value 为 255，将源图像像素转换为 RGB565 格式，并写入帧缓冲区。
   - 如果 opacity_value 介于 0 和 255 之间，执行 Alpha 混合操作将源图像像素与对应的帧缓冲区像素进行混合。混合公式为 ((255 - Sa) * D + Sa * S) / 255)，将混合结果写入帧缓冲区。

7.4.4 无 RLE 滤黑模式概述

以下流程描述了 No RLE 压缩图像的 Filter mode 处理过程。根据图像矩阵和显示设备的像素字节数选择处理方法，并将其写入帧缓冲区。

图 11: Filter_mode_path

无 RLE 滤黑模式（不带矩阵变换）

下面的流程图描述了将 uncompressed images 写入帧缓冲区的 filter mode 处理过程，以 RGB565 为目标设备图像类型为例。

图 12: Filter_blit_2_rgb565

1. 根据 img_type 执行不同的处理步骤。
2. 如果像素值为 0，则跳过处理；否则，执行后续写入操作。
3. 基于 opacity_value 执行相应的操作将图像像素写入帧缓冲区。
   - 如果 opacity_value 为 0，表示图像不显示，直接跳出处理流程。
   - 如果 opacity_value 为 255，将源图像像素转换为 RGB565 格式，并写入帧缓冲区。
   - 如果 opacity_value 介于 0 和 255 之间，执行 Alpha 混合操作将源图像像素与对应的帧缓冲区像素进行混合。混合公式为 ((255 - Sa) * D + Sa * S) / 255)，将混合结果写入帧缓冲区。

无 RLE 旁路模式（带矩阵变换）

下面的流程图描述了使用 Filter mode with matrix operations 将 Uncompressed images 写入帧缓冲区的过程，以 RGB565 为目标设备图像类型为例。

图 13: Filter_matrix_blit_2_rgb565

1. 根据 img_type 执行不同的处理步骤。
2. 执行矩阵计算，将目标区域的写入点映射到图像像素，并获取图像像素的像素值。
3. 如果像素值为 0，则跳过处理；否则，执行后续写入操作。
4. 基于 opacity_value 执行相应的操作将图像像素写入帧缓冲区。

- 如果 opacity_value 为 0，表示图像不显示，直接跳出处理流程。
- 如果 opacity_value 为 255，将源图像像素转换为 RGB565 格式，并写入帧缓冲区。
- 如果 opacity_value 介于 0 和 255 之间，执行 Alpha 混合操作将源图像像素与对应的帧缓冲区像素进行混合。混合公式为 ((255 - Sa) * D + Sa * S) / 255)，将混合结果写入帧缓冲区。

7.4.5 无 RLE 混合模式概述

以下流程描述了 No RLE 压缩图像的 source_over mode 处理过程。根据图像矩阵和显示设备的像素字节数选择处理方法，并将其写入帧缓冲区。

图 14: Alpha_mode_path

无 RLE 混合覆盖模式（不带矩阵变换）

下面的流程图描述了将 Uncompressed images 写入帧缓冲区的 Source_over mode 处理过程，以 RGB565 为目标设备图像类型为例。

图 15: Alpha_blit_2_rgb565

基于 opacity_value 执行相应的操作将图像像素写入帧缓冲区。- 如果 opacity_value 为 0，表示图像不显示，直接跳出处理流程。- 如果 opacity_value 为 255，将源图像像素转换为 RGB565 格式，并写入帧缓冲区。- 如果 opacity_value 介于 0 和 255 之间，执行 do_blending_acc_2_rgb565_opacity 对源图像像素与相应的帧缓冲区像素进行混合，将混合结果写入帧缓冲区。

无 RLE 混合覆盖模式（带矩阵变换）

下面的流程图描述了使用 Source_over mode with matrix operations 将 Uncompressed images 写入帧缓冲区的过程，以 RGB565 为目标设备图像类型为例。

图 16: Alpha_matrix_blit_2_rgb565

1. 执行矩阵计算，将目标区域的写入点映射到图像像素，并获取图像像素的像素值。
2. 基于 opacity_value 执行相应的操作将图像像素写入帧缓冲区。

- 如果 opacity_value 为 0，表示图像不显示，直接跳出处理流程。
- 如果 opacity_value 为 255，将源图像像素转换为 RGB565 格式，并写入帧缓冲区。
- 如果 opacity_value 介于 0 和 255 之间，执行 do_blending_acc_2_rgb565_opacity 对源图像像素与相应的帧缓冲区像素进行混合，将混合结果写入帧缓冲区。

7.4.6 RLE 覆盖模式概述

以下流程描述了 RLE 压缩图像的 Cover mode 处理过程。根据图像矩阵和显示设备的像素字节数选择处理方法，并将其写入帧缓冲区。

图 17: Rle_cover_mode_path

RLE 覆盖模式（不带矩阵变换）

下面的流程图描述了将 Compressed images 写入帧缓冲区的 Cover mode 处理过程，以 RGB565 为目标设备图像类型为例。

图 18: Rle_cover_blit_2_rgb565

1. 根据压缩数据头部的 img_type 执行不同的处理步骤。
2. 对压缩图像数据进行解压。
3. 将像素结果写入帧缓冲区。

RLE 覆盖模式（带矩阵变换）

下面的流程图描述了使用 Cover mode with matrix operations 将 Compressed images 写入帧缓冲区的过程，以 RGB565 为目标设备图像类型为例。

图 19: Rle_cover_matrix_blit_2_rgb565

1. 根据压缩数据头部的 img_type 执行不同的处理步骤。
2. 对压缩图像数据进行解压。
3. 进行矩阵计算，将目标区域的写入点映射到图像像素，并获得图像像素的像素值。
4. 将像素结果写入帧缓冲区。

7.4.7 RLE 旁路模式概述

以下流程描述了 RLE 压缩图像的 Bypass mode 处理过程。根据图像矩阵和显示设备的像素字节数选择处理方法，并将其写入帧缓冲区。

图 20: Rle_bypass_mode_path

RLE 旁路模式（不带矩阵变换）

下面的流程图描述了将 Bypass images 写入帧缓冲区的 Cover mode 处理过程，以 RGB565 为目标设备图像类型为例。

图 21: Rle_bypass_blit_2_rgb565

1. 根据压缩数据头部的 img_type 执行不同的处理步骤。
2. 对压缩图像数据进行解压。
3. 基于 opacity_value 执行相应的操作将图像像素写入帧缓冲区。

- 如果 opacity_value 为 0，表示图像不显示，直接跳出处理流程。
- 如果 opacity_value 为 255，将源图像像素转换为 RGB565 格式，并写入帧缓冲区。
- 如果 opacity_value 介于 0 和 255 之间，执行 Alpha 混合操作将源图像像素与对应的帧缓冲区像素进行混合。混合公式为 ((255 - Sa) * D + Sa * S) / 255)，将混合结果写入帧缓冲区。

RLE 旁路模式（带矩阵变换）

下面的流程图描述了使用 Bypass mode with matrix operations 将 Compressed images 写入帧缓冲区的过程，以 RGB565 为目标设备图像类型为例。

图 22: Rle_bypass_matrix_blit_2_rgb565

1. 根据压缩数据头部的 img_type 执行不同的处理步骤。
2. 对压缩图像数据进行解压。
3. 进行矩阵计算，将目标区域的写入点映射到图像像素，并获得图像像素的像素值。
4. 基于 opacity_value 执行相应的操作将图像像素写入帧缓冲区。

- 如果 opacity_value 为 0，表示图像不显示，直接跳出处理流程。
- 如果 opacity_value 为 255，将源图像像素转换为 RGB565 格式，并写入帧缓冲区。
- 如果 opacity_value 介于 0 和 255 之间，执行 Alpha 混合操作将源图像像素与对应的帧缓冲区像素进行混合。混合公式为 ((255 - Sa) * D + Sa * S) / 255)，将混合结果写入帧缓冲区。

7.4.8 RLE 滤黑模式概述

以下流程描述了 RLE 压缩图像的 Filter mode 处理过程。根据图像矩阵和显示设备的像素字节数选择处理方法，并将其写入帧缓冲区。

图 23: Rle_filter_mode_path

RLE 滤黑模式（不带矩阵变换）

下面的流程图描述了将 Compressed images 写入帧缓冲区的 Filter mode 处理过程，以 RGB565 为目标设备图像类型为例。

图 24: Rle_filter_blit_2_rgb565

1. 根据压缩数据头部的 img_type 执行不同的处理步骤。
2. 对压缩图像数据进行解压。
3. 如果像素值为 0 ，则跳过处理；否则，执行后续写入操作。
4. 基于 opacity_value 执行相应的操作将图像像素写入帧缓冲区。

- 如果 opacity_value 为 0，表示图像不显示，直接跳出处理流程。
- 如果 opacity_value 为 255，将源图像像素转换为 RGB565 格式，并写入帧缓冲区。
- 如果 opacity_value 介于 0 和 255 之间，执行 Alpha 混合操作将源图像像素与对应的帧缓冲区像素进行混合。混合公式为 ((255 - Sa) * D + Sa * S) / 255)，将混合结果写入帧缓冲区。

RLE 滤黑模式（带矩阵变换）

下面的流程图描述了使用 Filter mode with matrix operations 将 Compressed images 写入帧缓冲区的过程，以 RGB565 为目标设备图像类型为例。

图 25: Rle_filter_matrix_blit_2_rgb565

1. 根据压缩数据头部的 img_type 执行不同的处理步骤。
2. 对压缩图像数据进行解压。
3. 进行矩阵计算，将目标区域的写入点映射到图像像素，并获得图像像素的像素值。
4. 如果像素值为 0 ，则跳过处理；否则，执行后续写入操作。
5. 基于 opacity_value 执行相应的操作将图像像素写入帧缓冲区。

- 如果 opacity_value 为 0，表示图像不显示，直接跳出处理流程。
- 如果 opacity_value 为 255，将源图像像素转换为 RGB565 格式，并写入帧缓冲区。
- 如果 opacity_value 介于 0 和 255 之间，执行 Alpha 混合操作将源图像像素与对应的帧缓冲区像素进行混合。混合公式为 ((255 - Sa) * D + Sa * S) / 255)，将混合结果写入帧缓冲区。

7.4.9 RLE 混合模式概述

以下流程描述了 RLE 压缩图像的 source_over mode 处理过程。根据图像矩阵和显示设备的像素字节数选择处理方法，并将其写入帧缓冲区。

图 26: Rle_alpha_mode_path

RLE 混合模式（不带矩阵变换）

下面的流程图描述了将 Compressed images 写入帧缓冲区的 source_over mode 处理过程，以 RGB565 为目标设备图像类型为例。

图 27: Rle_alpha_blit_2_rgb565

1. 根据压缩数据头部的 img_type 执行不同的处理步骤。
2. 对压缩图像数据进行解压。
3. 基于 opacity_value 执行相应的操作将图像像素写入帧缓冲区。

- 如果 opacity_value 为 0，表示图像不显示，直接跳出处理流程。
- 如果 opacity_value 为 255，当源图像为 RGB565 格式时，直接将其写入帧缓冲区。否则，执行相应的混合操作 Do blend ，并将混合结果写入帧缓冲区。
- 如果 opacity_value 介于 0 和 255 之间，执行适当的混合操作 do_blending 来混合源图像像素与相应的帧缓冲区像素，将混合结果写入帧缓冲区。

RLE 源覆盖模式（带矩阵变换）

下面的流程图描述了使用 Source_over mode with matrix operations 将 Compressed images 写入帧缓冲区的过程，以 RGB565 为目标设备图像类型为例。

图 28: Rle_alpha_matrix_blit_2_rgb565

1. 根据压缩数据头部的 img_type 执行不同的处理步骤。
2. 对压缩图像数据进行解压。
3. 进行矩阵计算，将目标区域的写入点映射到图像像素，并获得图像像素的像素值。
4. 基于 opacity_value 执行相应的操作将图像像素写入帧缓冲区。

- 如果 opacity_value 为 0，表示图像不显示，直接跳出处理流程。
- 如果 opacity_value 为 255，当源图像为 RGB565 格式时，直接将其写入帧缓冲区。否则，执行相应的混合操作 Do blend ，并将混合结果写入帧缓冲区。
- 如果 opacity_value 介于 0 和 255 之间，执行适当的混合操作 do_blending 来混合源图像像素与相应的帧缓冲区像素，将混合结果写入帧缓冲区。

备注： 在压缩的 source_over 矩阵模式下，rle_rgb888 和 rle_rgba8888 相当于输出 rle_rgb565。

7.4.10 支持的输入类型和输出类型

输入类型	输出类型
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 8

常见问题

使用 GUI 过程中产生的一些常见问题，可参考本章节。

8.1 开发环境

8.1.1 在 VSCode 中使用模拟器

如果您首次在 VSCode 中运行模拟器时遇到问题，请检查开发环境中的以下配置：

安装适当版本的工具链

在 VSCode 中使用 MinGW 工具链进行模拟器开发，推荐使用 MinGW 版本 8.1.0，您可以从 MinGW v8.1.0 Download 下载获取。不能保证所有更新版本的 MinGW 能够正常工作。

> **警告：** VSCode 目前不支持使用高于 v9.2.0 的 MinGW 中的 gdb 版本。（推荐使用 MinGW v8.1.0 中的 gdb v8.1）

添加工具链到系统环境变量

请确保 `C:/mingw64/bin` 已添加到系统环境变量 `Path` 中。

8.2 移植

8.2.1 用户数据

生成用户数据 bin 镜像时，需要考虑 `flash_map.h` 中的用户数据地址。通常，生成脚本中的地址与 `flash_map.h` 中的用户数据地址一致。如果由于 mppgtool 要求，用户数据 bin 需要添加图像数据头，则生成脚本中的地址必须增加图像数据头大小。

8.2.2 JS 分配堆内存

JS（JavaScript）包含在 GUI 模块中，JS 使用的堆空间可能会受到资源限制，导致 `malloc` 失败。如果 SoC 支持 psram 特性，可以将此堆空间重定位到 psram。具体信息请参考 API `void *context_alloc(size_t size, void *cb_data_p)`。

8.2.3 喂狗

GUI 任务不支持喂狗功能，因此应用程序应在 APP 注册的 hook 函数中进行喂狗操作，并由 GUI 使用。注册函数为 `void gui_task_ext_execution_sethook(void (*hook)(void))`。

8.2.4 不支持 FPU

如果 SoC 不支持 FPU，一些头文件和代码不应该包含在内，例如 RTL8763EP。

8.2.5 文件系统

SoC 需要通过文件系统从 flash 中读取图像和字体资源，文件系统中的起始地址应与用户数据生成脚本中的地址一致。GUI 已经提供了相关文件，其中起始地址为 `ROMFS_ADDR`。

8.2.6 Flash 设置

Flash 速度模式应设置为 4 位模式；根据芯片的能力，Flash 时钟应设置为更高的值。

8.2.7 CPU 频率

根据芯片的能力，CPU 频率应设置为更高的值。

8.2.8 SCONS 版本

需要特定的 SCONS 版本，请使用 `pip install scons==4.4.0` 命令进行下载。

8.3 规格

8.3.1 图形

平台	RTL8762D	RTL8772F	RTL87X2G	RTL8763E	PC
RGB565	是	是	是	是	是
RGB888	否	是	是	否	是
ARGB8888	否	是	是	否	是
SVG	否	是	否	否	是
TTF	否	是	否	否	是
DOT 字体	是	是	是	是	是
矢量图形	否	是	否	否	是
线性渐变	否	是	否	否	是
径向渐变	否	否	否	否	是

8.3.2 内存使用量

RTL8772F 示例

该示例的内存消耗统计如下：

模块	占用
控件	14.56KB
帧缓冲区	屏幕宽度 * 像素字节数 * 行数
线程堆栈	10KB

控件内存使用量

控件	内存 (字节) 系统芯片	内存 (字节) 模拟器
对象 (obj)	52	88
图像 (img)	112	178
窗口 (win)	72	112
页面 (page)	124	184
选项卡 (tab)	88	136
选项卡视图 (tabview)	100	160
按钮 (button)	88	160
文本框 (text)	100	176
滚动文本框 (scroll_text)	120	200
应用 (app)	92	152
画布-圆弧 (canvas_arc)	156	264
画布-矩形 (canvas_rect)	64	104
画布 (canvas)	60	104
卡片 (card)	72	112
卡片视图 (cardview)	124	176
调色盘 (colorwheel)	72	112

续下页

表 1 – 接上页

控件	内存 (字节) 系统芯片	内存 (字节) 模拟器
立方体 (cube)	748	928
窗帘 (curtain)	60	96
窗帘视图 (curtainview)	120	168
画廊 (gallery)	112	184
网格 (grid)	100	144
动态图像 (img_live)	84	144
范围图像 (img_scope)	124	192
stb 图像 (stb_img)	76	144
键盘 (kb)	108	192
地图 (map)	196	272
蜂窝菜单 (menu_cellular)	76	120
多级菜单 (multi_level)	60	104
页面列表 (pagelist)	96	160
页面列表视图 (pagelistview)	64	112
透视 (perspective)	736	920
进度条 (progressbar)	80	136
二维码和条形码 (qbcode)	84	136
滚轮 (scroll_wheel)	388	696
进度条 (seekbar)	128	216
简单图像 (simple_img)	68	120
可缩放矢量图形 (svg)	96	144
轮盘 (turn_table)	128	192
监视渐变点 (watch_gradient_spot)	60	96
波 (wave)	72	112
轮形列表 (wheel_list)	64	112

8.4 帧率

8.4.1 像素格式

使用 RGBA/RGB 图像可以获得出色的显示效果，但如果帧率较低，可以使用 RGB565 格式的图像资源，以稍微牺牲一些效果来提高帧率。

8.4.2 硬件加速

尽可能使用硬件加速来渲染图像，而不是软件加速。不同的芯片型号可能具有不同的 GPU，请参考 SDK 中的指导文档获取详细信息。

8.4.3 数据传输速度

数据传输速度也会影响 HoneyGUI 的绘制速度，因此请确保内存（FLASH 和 PSRAM）的带宽和频率。

8.4.4 UI 设计

减少界面中的复杂性，以及单个界面中图像的数量和大小可以提高帧率。确保每个需要加载的图像数据的像素都是有用的。

8.4.5 图像压缩

HoneyGUI 支持图像压缩，并且部分芯片内置硬件解压缩模块。使用硬件解压缩模块是非常快的，但是使用软件解压缩就需要一定的时间。使用压缩图像可以降低原始图像资源的大小，可以在 `userrdata` 中放置更多的资源，并且会减少读取 flash 的时间。

8.4.6 字体

自定义二进制文件

- 尽可能使用字体大小为 8 的倍数。
- 当文件包含数百个字符时，在创建字体文件时应将 `indexMethod` 设置为 0。

标准 TTF 文件

- 使用 TTF 文件来显示文本比使用 BIN 文件要慢得多。
- TTF 文件可以通过开源解决方案进行裁剪。

8.5 显示

8.5.1 字体抗锯齿

- 字体抗锯齿效果差且白色字体边缘有异常彩色

  在使用 2bit 以上的字库时，如果字体抗锯齿效果差，且字体边缘有异常彩色或者字体颜色显示异常，可能是字体渲染数据的大小端问题。请通过显示 RGB 单通道色彩字体来判别，比如设置字体颜色为 `gui_rgb(255,0,0,255)`，正常情况会显示红色，若颜色是蓝色则异常（彩色字体都能看出异常）。

CHAPTER 9

获取 PDF

PDF 版本: RTKIOT GUI.pdf

CHAPTER 10

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 11

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)

索引

I

K

L

M

O

P

R

T

V