GR5xx APP Log Application Note

Version: 3.2

Release Date: 2023-11-06

Shenzhen Goodix Technology Co., Ltd.

Shenzhen Goodix Technology Co., Ltd.

Headquarters: Floor 12-13, Phase B, Tengfei Industrial Building, Futian Free Trade Zone, Shenzhen, China

TEL: +86-755-33338828 Zip Code: 518000

Website: www.goodix.com

Preface

Purpose

This document introduces the functionalities, operating mechanisms, and applications of APP Log module in Bluetooth Low Energy (Bluetooth LE) GR5xx Software Development Kit (SDK), to help developers quickly get started with secondary development of the module.

Audience

This document is intended for:

- Device user
- Developer
- Test engineer
- Hobbyist developer

Release Notes

This document is the fourth release of *GR5xx APP Log Application Note*, corresponding to Bluetooth LE GR5xx System-on-Chip (SoC) series.

Revision History

Version	Date	Description
1.0	2022-05-10	Initial release
3.0	2023-03-30	• Updated descriptions about GR5xx SoCs. • Updated the code in sections "Log Output" and "Log Storage and Export".
3.1	2023-08-08	• Updated the file directory in "Adding Source Files". • Updated the descriptions and code in "Log Output" and "Log Storage and Export".
3.2	2023-11-06	Updated the approaches for obtaining GRToolbox and GRUart.

Contents

1 Introduction

GR5xx APP Log module is provided in GR5xx Software Development Kit (SDK) to assist developers in development and debugging, supporting the following functionalities:

- Output logs in real time. You can customize the output mode of debug logs (through a hardware port such as UART or J-Link RTT).
- Store and export logs. You can store the logs in Flash of GR5xx System-on-Chips (SoCs), and obtain the logs on the mobile App GRToolbox (Android) through Bluetooth connection when needed.
- Set log levels and filter logs. You can output logs at multiple levels (DEBUG, INFO, WARNING, ERROR) and filter logs by levels, to record information such as log level, time, and source.

Before getting started, you can refer to the following documents.

Table 1-1 Reference documents

Name	Description
Developer guide of the specific GR5xx SoC	Introduces GR5xx SDK and how to develop and debug applications based on the SDK.
J-Link/J-Trace User Guide	Provides J-Link operational instructions. Available at https://www.segger.com/downloads/jlink/UM08001_JLink.pdf.
Keil User Guide	Offers detailed Keil operational instructions. Available at https://www.keil.com/support/man/docs/uv4/.

2 Environment Setup

This chapter introduces how to rapidly set up an operating environment for GR5xx APP Log module.

2.1 Preparation

Perform the following tasks before applying GR5xx APP Log module.

- **Hardware preparation**

Table 2-1 Hardware preparation

Name	Description
Development board	Starter Kit Board (SK Board) of the corresponding SoC
Connection cable	USB Type C cable (Micro USB 2.0 cable for GR551x SoCs)
Android phone	A mobile phone running on Android 5.0 (KitKat) and later

- **Software preparation**

Table 2-2 Software preparation

Name	Description
Windows	Windows 7/Windows 10
J-Link driver	A J-Link driver. Available at https://www.segger.com/downloads/jlink/.
Keil MDK	An integrated development environment (IDE). MDK-ARM Version 5.20 or later is required. Available at https://www.keil.com/download/product/.
J-Link RTT Viewer (Windows)	A J-Link log output tool. Available at https://www.segger.com/products/debug-probes/j-link/tools/rtt-viewer/.
GRUart (Windows)	A serial port debugging tool. Available at https://www.goodix.com/en/download?objectId=43&objectType=software.
GRToolbox (Android)	A Bluetooth LE debugging tool. Available at https://www.goodix.com/en/software_tool/grtoolbox.

3 Application of APP Log Module

This chapter introduces how to add GR5xx APP Log module to a project and how to use the module by taking ble_app_pcs (an example project) in GR5xx SDK as an example.

3.1 Importing APP Log Module

APP Log module is optional for running a GR5xx-based project. Before using the module, add the files of APP Log module to the project directory and enable the macro switch of the module.

3.1.1 Adding Source Files

The ble_app_rscs and ble_app_template_freertos projects in GR5xx SDK enable log-related functionalities of APP Log module and implement log storage and export. You can refer to the two projects for porting and development.

The table below lists the source files of APP Log module.

Table 3-1 Source files of APP Log module

File	Description
SDK_Folder\components\libraries \app_log\app_log.c	Source file of APP Log module. It is required to add the file before using APP Log module.
SDK_Folder\components\libraries \app_log\app_log_store.c	Source file for log storage of APP Log module. It is required to add the file before using the log storage and export functionalities of APP Log module.
SDK_Folder\components\libraries \app_log\app_log_dump_port.c	Source file for exporting stored logs through Bluetooth. It is required to add the file before using the log storage and export functionalities of APP Log module.
SDK_Folder\components\profiles \lms\lms.c	Source file corresponding to Bluetooth service for log export. It is required to add the file before using the log storage and export functionalities of APP Log module.

Note:

SDK_Folder is the root directory of the GR5xx SDK in use.

The steps to add related source files of APP Log module are as follows by taking ble_app_pcs in GR5xx SDK as an example:

1. Run ble_app_pcs.

   The source code and project file of ble_app_pcs are in `SDK_Folder\projects\ble\ble_peripheral\ble_app_pcs`, and project file is in the Keil_5 folder.

2. Add the source files of APP Log module to the project directory of ble_app_pcs.

   (1). Select and right-click **GRxx_Soc**, and then choose **Add Group** to add a directory named as "gr_board". Select and right-click **gr_board**, and then choose **Add Existing Files to Group 'gr_board'** to add the file in `SDK_Folder\platform\boards\board_SK.c`.

(2). Select and right-click **gr_libraries**. Choose **Add Existing Files to Group 'gr_libraries'** to add *app_error.c*, *app_assert.c*, *app_log.c*, *app_log_store.c*, and *app_log_dump_port.c* to **gr_libraries**, as shown in Figure 3-1.

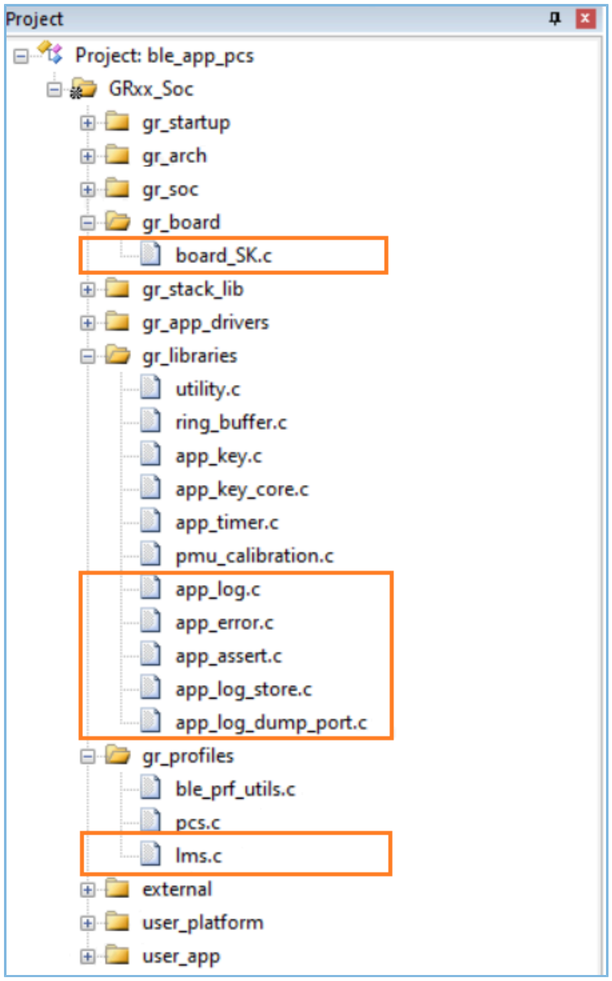

Figure 3-1 Adding source files into the project

(3). Select and right-click **gr_profiles**. Choose **Add Existing Files to Group 'gr_profiles'** to add *lms.c* to gr_profiles, and add the corresponding header file path, as shown below:

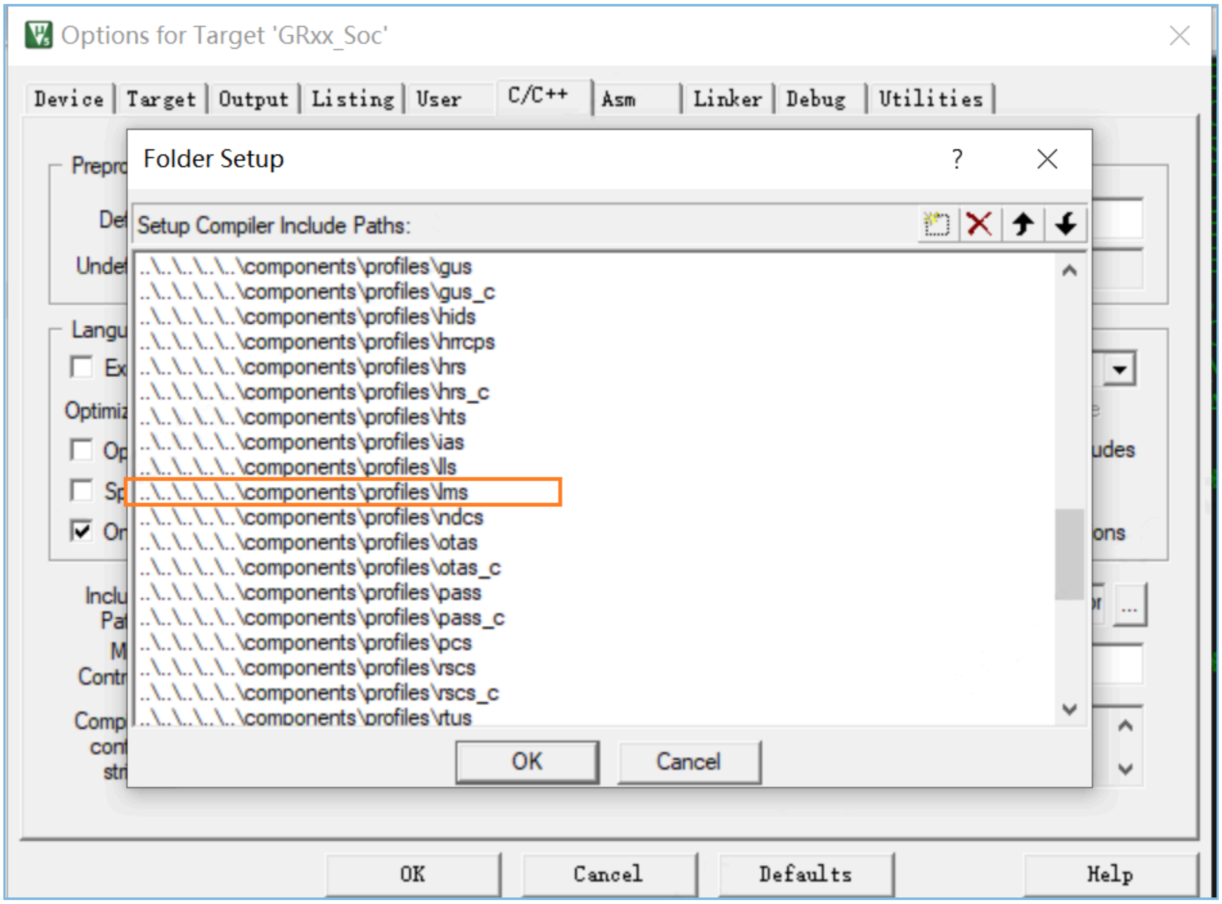

Figure 3-2 Adding header files into the project

According to the output port adopted for the APP Log module, the UART driver source file and SEGGER RTT source driver file may be needed, depending on the configured output mode. The steps to add the two files are similar to those to add the sources files of APP Log module.

Currently, the two files have been added to all projects in GR5xx SDK by default.

- The UART driver source file is in `SDK_Folder\drivers\src`.
- The SEGGER RTT driver source file is in `SDK_Folder\external\segger_rtt`.

3.1.2 Configuring Mode and Functionality

Macros related to APP Log module are defined in *custom_config.h*, as shown below. You can configure the mode and functionalities of APP Log module according to project requirements and hardware environment.

```
// <o> Enable APP log module
// <0=> DISABLE
// <1=> ENABLE
#ifndef APP_LOG_ENABLE
#define APP_LOG_ENABLE           1
#endif

// <o> APP log port type
// <0=> UART
// <1=> RTT
```



```
// <2=> ITM
#ifndef APP_LOG_PORT
#define APP_LOG_PORT             0
#endif

// <o> Enable APP log store module
// <0=> DISABLE
// <1=> ENABLE
#ifndef APP_LOG_STORE_ENABLE
#define APP_LOG_STORE_ENABLE     0
#endif
```

Table 3-2 Macro description of APP Log module

Macro	Definition
APP_LOG_ENABLE	Enable/Disable APP Log module. • 0: Disable APP Log module. • 1: Enable APP Log module.
APP_LOG_PORT	Set the output mode of APP Log module. • 0: UART • 1: J-Link RTT • 2: ITM
APP_LOG_STORE_ENABLE	Enable/Disable the log storage functionality of APP Log module. • 0: Disable the log storage functionality. • 1: Enable the log storage functionality.

3.2 Module Initialization and Scheduling

After configuration, you need to call related initialization function during peripheral initialization to complete the initialization, and call related scheduling function when appropriate. The initialization and scheduling functions to be called vary according to the specific App Log functionalities required. The sections below introduce the application and scenarios of related APIs.

3.2.1 Log Output

If only the log output functionality is required, you can call app_log_init() of APP Log module to complete module initialization.

The input parameters of app_log_init() include the log initialization parameter, log output API, and flush API (optional for registration). Call the initialization function of corresponding API and register corresponding the transmission and flush functions according to the configured output port.

- To output debug logs through UART port, UART-related initialization function shall be called. Taking *board_SK.c* as an example, bsp_uart_init (UART initialization function), bsp_uart_send (UART transmission function), and bsp_uart_flush (UART flush function) shall be executed to initialize APP Log module. The code snippet is as follows:

Note:

board_SK.c is in `SDK_Folder\platform\boards\board_SK.c`.

```
void bsp_log_init(void)
{
#if (APP_LOG_ENABLE == 1)
#if (APP_LOG_PORT == 0)
    bsp_uart_init();
#elif (APP_LOG_PORT == 1)
    SEGGER_RTT_ConfigUpBuffer(0, NULL, NULL, 0, SEGGER_RTT_MODE_NO_BLOCK_TRIM);
#endif
#if (APP_LOG_PORT <= 2)
    app_log_init_t   log_init;
    log_init.filter.level                  = APP_LOG_LVL_DEBUG;
    log_init.fmt_set[APP_LOG_LVL_ERROR]    = APP_LOG_FMT_ALL & (~APP_LOG_FMT_TAG);
    log_init.fmt_set[APP_LOG_LVL_WARNING] = APP_LOG_FMT_LVL;
    log_init.fmt_set[APP_LOG_LVL_INFO]     = APP_LOG_FMT_LVL;
    log_init.fmt_set[APP_LOG_LVL_DEBUG]    = APP_LOG_FMT_LVL;
#if (APP_LOG_PORT == 0)
    app_log_init(&log_init, bsp_uart_send, bsp_uart_flush);
#elif (APP_LOG_PORT == 1)
    app_log_init(&log_init, bsp_segger_rtt_send, NULL);
#elif (APP_LOG_PORT == 2)
    app_log_init(&log_init, bsp_itm_send, NULL);
#endif
    app_assert_init();
#endif
#endif
}
```

Related parameters are described as follows:

  - bsp_uart_send is to implement app_uart async (app_uart_transmit_async API) and hal_uart sync (hal_uart_transmit API) output APIs. You can select a proper log output mode according to specific application requirements.
  - bsp_uart_flush is a uart_flush API for outputting the remaining data cached in RAM of GR5xx SoCs in interrupt mode.

You can rewrite the above two APIs.

- When debug logs are output through J-Link RTT port, the implemented log output API is bsp_segger_rtt_send(). No flush API is to be implemented in this mode.

  Initialization of different output modes has been implemented in *board_SK.c*. When using *board_SK.c* directly, you only need to configure APP_LOG_PORT to select the log output mode. You can also refer to *board_SK.c* for development.

If asynchronous output mode is adopted (such as asynchronous output in interrupt mode through UART port), app_log_flush() shall be called in scenarios where cached data needs to be cleared, to output all logs in the cache to prevent logs from missing due to cache clearing. For example, app_log_flush() shall be called before the system enters sleep mode. The code snippet is as follows:

```
…
#include "app_log.h"
```



```
…
int main(void)
{
    // Initialize user peripherals.
    app_periph_init();

    if (is_enter_ultra_deep_sleep())
    {
        pwr_mgmt_ultra_sleep(0);
    }

    // Initialize ble stack.
    ble_stack_init(ble_evt_handler, &heaps_table);

    // Loop
    while (1)
    {
        app_log_flush();
        pwr_mgmt_schedule();
    }
}
```

app_log_flash() calls the flush API registered by users during initialization to implement all output functionalities.

3.2.2 Log Storage and Export

To use the log storage and export functionalities, you need to call app_log_store_init() to complete log storage-related configurations, and initialize the log storage and export functionalities in `SDK_Folder\projects\ble\ble_peripheral\ble_app_pcs\Src\platform\user_periph_setup.c` for ble_app_pcs. The code snippet is as follows:

```
...
#include "board_SK.h"
#include "app_assert.h"
#include "app_log.h"
#include "flash_scatter_config.h"
...
static void log_store_init(void)
{
    app_log_store_info_t store_info;
    app_log_store_op_t   op_func;

    store_info.nv_tag   = 0x40ff;
    store_info.db_addr  = FLASH_START_ADDR + 0x60000;
    store_info.db_size  = 0x20000;
    store_info.blk_size = 0x1000;

    op_func.flash_init  = hal_flash_init;
    op_func.flash_erase = hal_flash_erase;
    op_func.flash_write = hal_flash_write;
    op_func.flash_read  = hal_flash_read;
    op_func.time_get    = NULL;
    op_func.sem_give    = NULL;
    op_func.sem_take    = NULL;

    app_log_store_init(&store_info, &op_func);
}
```

Structures in app_log_store_init() are described below:

- app_log_store_info_t: Contains information about log storage area; parameters involved include NVDS tag, start address for storage, storage area size, and storage area block size (minimum erasing unit).
- app_log_store_op_t: Contains operating functions and other functionality functions of Flash that stores the logs. All operating functions shall be implemented, including initialization, erasing, read, and write functions. Other functionality functions can be implemented according to specific circumstances.
    - To add real time to the stored log, op_func.time_get shall be implemented.
    - To use APP Log module in an environment equipped with an operating system, op_func.sem_give and op_func.sem_take shall be implemented.

Tip:

You can determine the initialization parameters of the module according to Flash layout and category of the operating system.

You also need to call log_store_init() and board_init() in app_periph_init(). The code snippet is as follows:

```
void app_periph_init(void)
{
   app_scheduler_init(APP_SCHEDULER_QUEUE_SIZE);
   SYS_SET_BD_ADDR(s_bd_addr);
   board_init();
#if APP_LOG_STORE_ENABLE
   log_store_init();
#endif
   pwr_mgmt_mode_set(PMR_MGMT_SLEEP_MODE);
}
```

Log storage and export shall be implemented in app_log_store_schedule(). Therefore, you shall call app_log_store_schedule() when needed.

- In ble_app_pcs, you need to call app_log_store_schedule() in main() loop, and comment out the code used for entering ultra-low power mode. The code snippet is as follows:

```
…
#include "app_log.h"
…
int main(void)
{
    // Initialize user peripherals.
    app_periph_init();
//    if (is_enter_ultra_deep_sleep())
//    {
//        pwr_mgmt_ultra_sleep(0);
//    }

    // Initialize ble stack.
    ble_stack_init(ble_evt_handler, &heaps_table);

    // Loop
    while (1)
    {
  app_log_flush();
  app_log_store_schedule();
        pwr_mgmt_schedule();
```

```
    }
}
```

- To use APP Log module in an environment equipped with an operating system, it is recommended to call app_log_store_schedule() (at low priority) independently, and signal amount-related APIs shall be registered during initialization (refer to ble_app_template_freertos). The scheduling mode is as follows:

```
static void log_store_dump_task(void *p_arg)
{
    while (1)
    {
        app_log_store_schedule();
    }
}
```

In addition, the log export functionality of APP Log module is implemented through Bluetooth transmission, so the Bluetooth service in use shall be initialized. It is recommended to call app_log_dump_service_init() in the callback function after initialization of the Bluetooth Low Energy (Bluetooth LE) Stack completes. In ble_app_pcs, you need to call app_log_dump_service_init() in services_init in *user_app.c*. The code snippet is as follows:

```
…
#include "app_log.h"
#include "app_log_dump_port.h"
…
static void services_init(void)
{
…
    app_log_dump_service_init();
…
}
```

Add print information into ble_app_init. The code snippet is as follows:

```
…
#include "app_error.h"
…
void ble_app_init(void)
{
    sdk_err_t         error_code;
    ble_gap_bdaddr_t  bd_addr;
    sdk_version_t     version;

    sys_sdk_verison_get(&version);
    APP_LOG_INFO("Goodix BLE SDK V%d.%d.%d (commit %x)",
                 version.major, version.minor, version.build, version.commit_id);

    error_code = ble_gap_addr_get(&bd_addr);
    APP_ERROR_CHECK(error_code);
    APP_LOG_INFO("Local Board %02X:%02X:%02X:%02X:%02X:%02X.",
                 bd_addr.gap_addr.addr[5],
                 bd_addr.gap_addr.addr[4],
                 bd_addr.gap_addr.addr[3],
                 bd_addr.gap_addr.addr[2],
                 bd_addr.gap_addr.addr[1],
                 bd_addr.gap_addr.addr[0]);
    APP_LOG_INFO("PCS example started.");
    …
```

```
}
```

You can use APP Log APIs to output debug logs (refer to "Section 3.3 Outputting Logs", which will be stored in Flash, and then you can export logs through GRToolbox (for details, refer to "Section 3.4 Obtaining Logs").

After modification (adding/enabling/initializing APP Log module) to a project, you can program the compiled project to the SK Board.

Note:

You need to set APP_LOG_ENABLE and APP_LOG_STORE_ENABLE to 1 in `SDK_Folder\projects\ble\ble_peripheral\ble_app_pcs\Src\config\custom_config.h` to enable the log and storage sub-modules.

3.3 Outputting Logs

The APP Log module supports using printf() (a C standard library function) and APIs provided in APP Log module to output debug logs.

- To output debug logs using printf(), set app_log_init_t *p_log_init in app_log_init() to "NULL". However, you cannot optimize logs by setting log level, log format, and filter type in APP Log module, and logs output in this way cannot be stored and exported.
- To output debug logs using APP Log APIs, you can call any of the following four APIs to output debug logs after initialization of the APP Log module:
    - APP_LOG_ERROR()
    - APP_LOG_WARNING()
    - APP_LOG_INFO()
    - APP_LOG_DEBUG()

  You can also optimize output logs by setting log level, log format, filter type, or other parameters, to further simplify application debugging.

Note:

You can set the log level and log filter type respectively by configuring APP_LOG_TAG and APP_LOG_SEVERITY_LEVEL in `SDK_Folder\components\libraries\app_log\app_log.h`.

3.4 Obtaining Logs

Logs can be obtained in real time or exported through GRToolbox.

3.4.1 Obtaining Logs in Real Time

You can obtain debug logs through a proper PC tool on a PC according to the configured output mode.

- To output logs through UART port, GRUart in GR5xx SDK can be used to obtain logs in real time.

  Connect the PC with the SK Board that you wish to read debug logs from, and start GRUart on the PC. After configuration completes, you can obtain debug logs from the SK Board, as shown below.

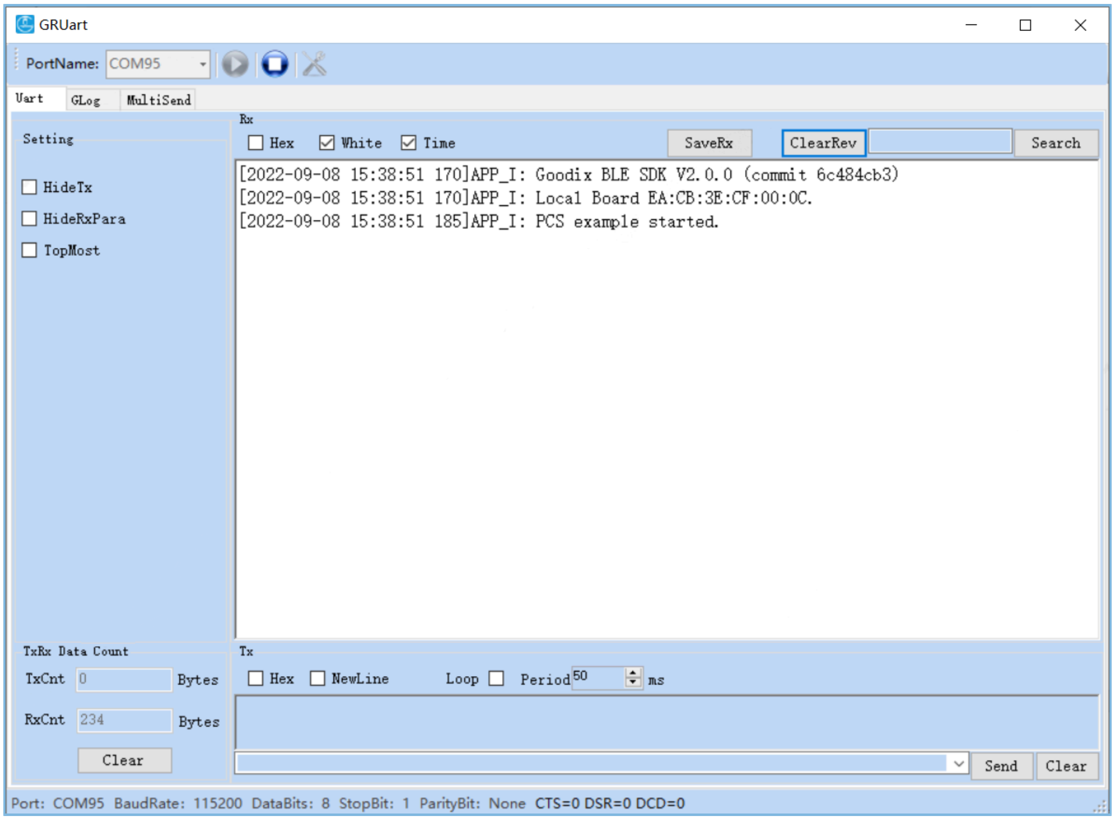

Figure 3-3 GRUart interface

- To output logs through J-Link RTT port, you can use J-Link RTT Viewer to obtain logs in real time.

  Connect the PC with the SK Board that you wish to read debug logs from, and start J-Link RTT Viewer on the PC to enter the configuration interface. Configure J-Link RTT Viewer as shown below.

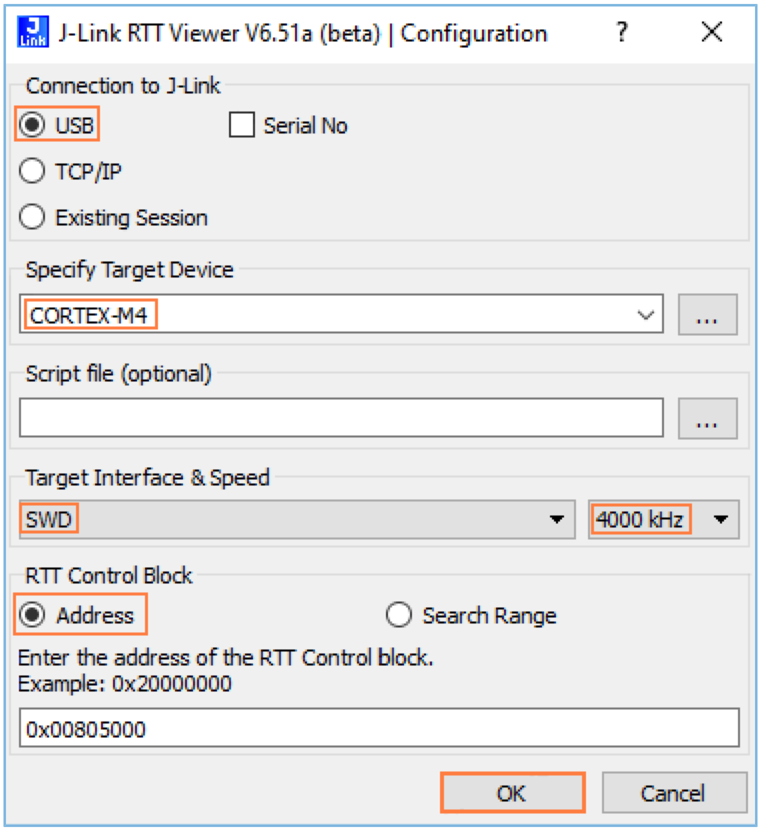

Figure 3-4 J-Link RTT Viewer configuration interface

Before configuring **RTT Control Block**, find out the address of **RTT Control Block** (the variable "_SEGGER_RTT").

- You can select **Search Range** in the J-Link RTT Viewer configuration interface and set the entire RAM address as the search range. Then J-Link RTT Viewer automatically searches the **RTT Control Block** address (not recommended due to slow search speed).
- You can also obtain the address by searching from the "_SEGGER_RTT" structure in the .map file generated by the project, and then select **Address** in the configuration interface to specify the **RTT Control Block** address.

It is recommended to modify *SEGGER_RTT.c* as follows to define **RTT Control Block** as the specified address, to improve efficiency. The code snippet for configuring **RTT Control Block** as **0x00805000** is as follows:

```
// RTT Control Block and allocate buffers for channel 0
//
__attribute__((section(".ARM. __at_0x00805000"))) SEGGER_RTT_CB _SEGGER_RTT
//SEGGER_RTT_PUT_CB_SECTION(SEGGER_RTT_CB_ALIGN(SEGGER_RTT_CB _SEGGER_RTT));
```


Note:

SEGGER_RTT.c is in `SDK_Folder\external\segger_rtt\SEGGER_RTT.c`.

After configuration completes, click **OK**. When the SK Board is connected with J-Link RTT Viewer, the J-Link RTT Viewer log interface will display, as shown below. Firmware logs shown in the interface indicates that the configuration succeeds.

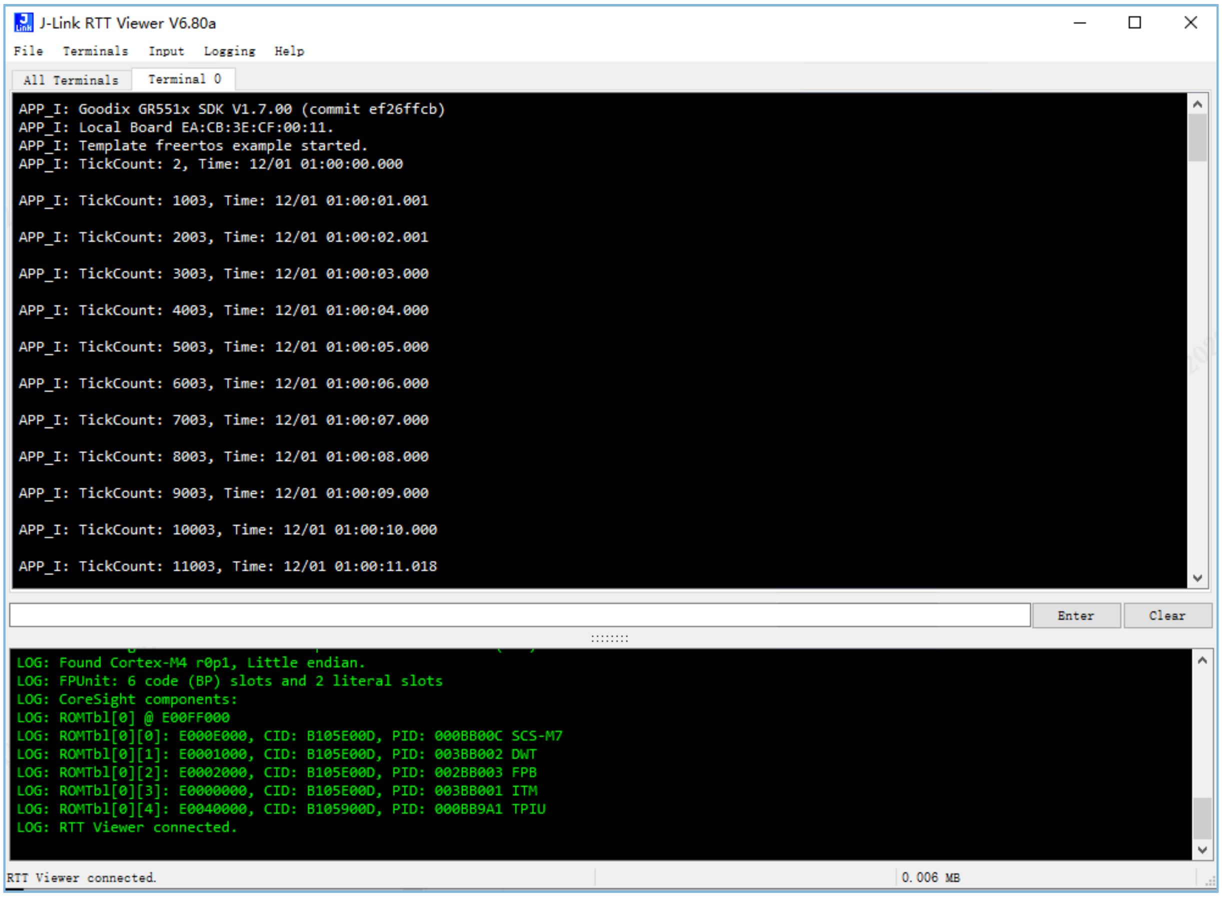

Figure 3-5 Log output interface of J-Link RTT Viewer

3.4.2 Exporting Stored Logs

GRToolbox (Android) in GR5xx SDK supports exporting logs in APP Log module.

The ble_app_template_freertos project is taken as an example to introduce the log export functionality (for detailed configurations, refer to "Section 3.1.2 Configuring Mode and Functionality").

1. Open GRToolbox on an Android phone and connect the phone with the SK Board. **Goodix Log Service** (GLS) is then discovered, as shown below.

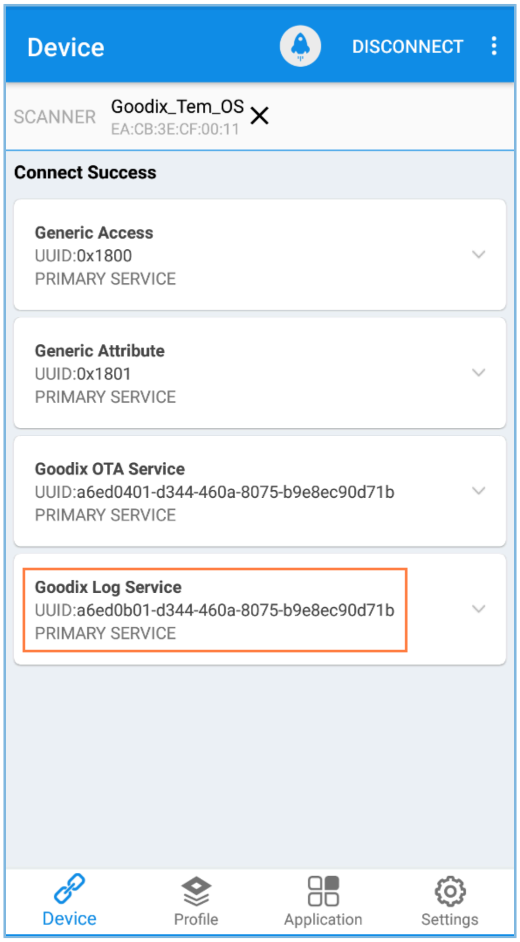

Figure 3-6 Successful discovery of GLS after connecting the phone to the Board through GRToolbox

---

Note:

GRToolbox screenshots in this document are used to help you better understand the operating steps only. The user interface of GRToolbox in actual use prevails.

---

2. Tap ⋮ in the upper-right corner and select **Dump Log** from the drop-down list:

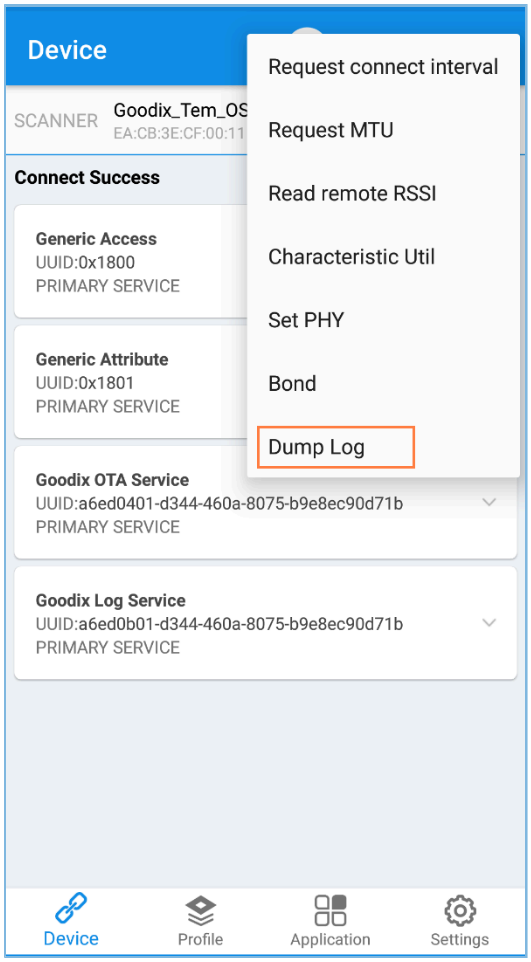

Figure 3-7 To output logs

3. In the **Dump Log** dialog box, you can delete/save/read logs, as shown below.

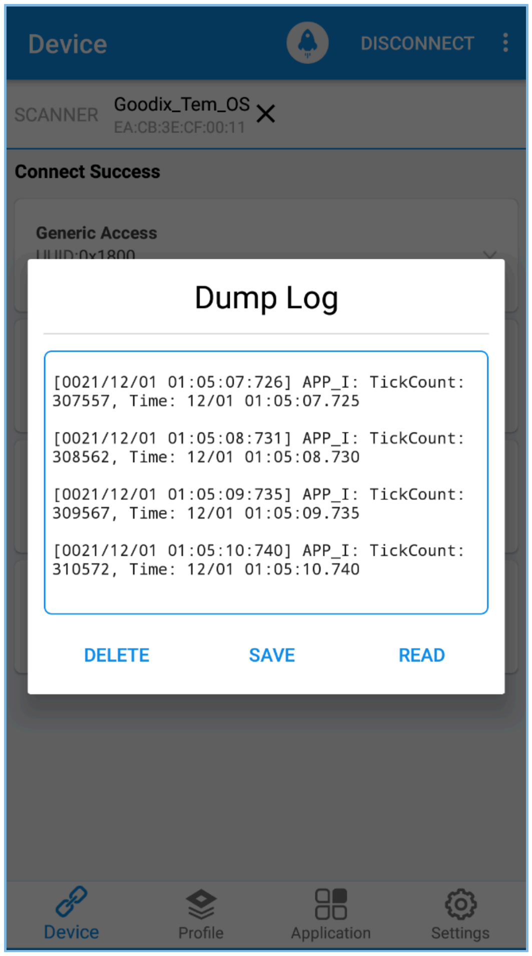

Figure 3-8 **Dump Log** interface of GRToolbox

4 Module Details

APP Log module provides log APIs at multiple levels. When you call these APIs, information such as log level, time, and source will be added to the beginning in original logs according to the API level, and logs will be filtered according to the filter type configured during initialization. Then logs will be transmitted by calling corresponding transmission function. The following figure shows the calling relationship between log output functions.

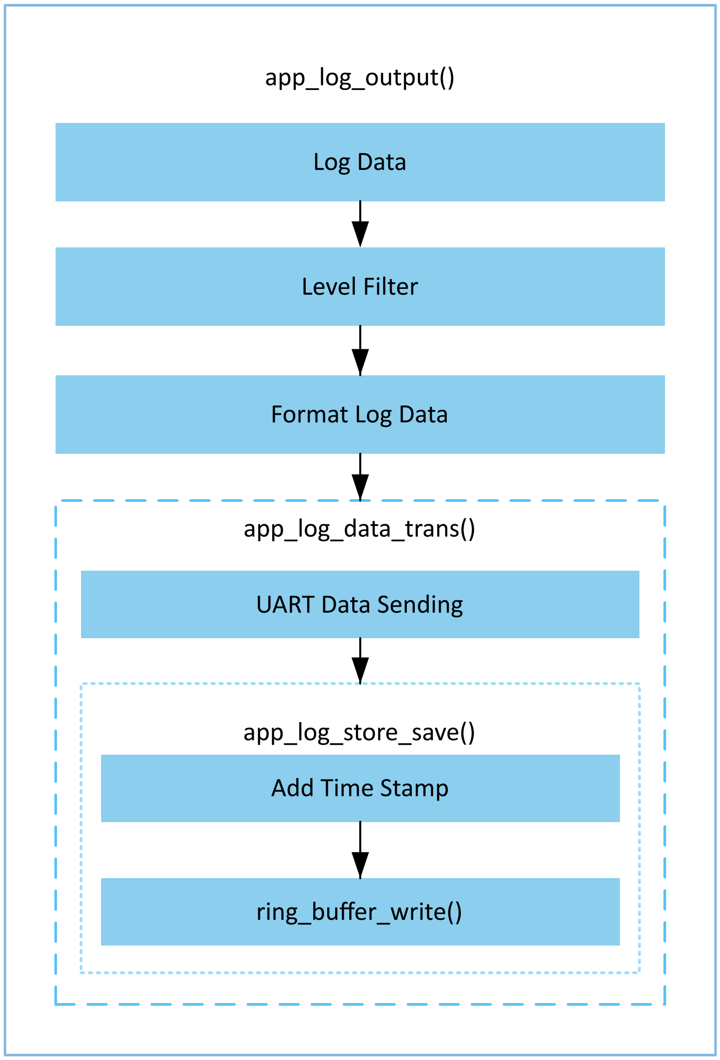

Figure 4-1 Calling relationship between log output functions

Note:

The logic code of APP Log module is in *app_log.c*.

4.1 Log Transmission and Storage APIs

Path: `gr_libraries\app_log.c` under the project directory

Name: app_log_data_trans()

```
static void app_log_data_trans(uint8_t *p_data, uint16_t length)
```



```
{
    if (NULL == p_data || 0 == length)
    {
        return;
    }

    if (s_app_log_env.trans_func)
    {
        s_app_log_env.trans_func(p_data, length);
    }

#if APP_LOG_STORE_ENABLE
    app_log_store_save(p_data, length);
#endif
}
```

Call s_app_log_env.trans_func (for example, UART transmission function) registered during module initialization in the transmission function, and determine whether to call app_log_store_save() based on whether APP_LOG_STORE_ENABLE is enabled.

Path: `gr_libraries\app_log_store.c` under the project directory

Name: app_log_store_save()

```
uint16_t app_log_store_save(const uint8_t *p_data, const uint16_t length)
{
    ...
    ring_buffer_write(&s_log_store_rbuf, time_encode, APP_LOG_STORE_TIME_SIZE);
    ring_buffer_write(&s_log_store_rbuf, p_data, length);
    if ((APP_LOG_STORE_ONECE_OP_SIZE <= ring_buffer_items_count_get(&s_log_store_rbuf)) &&
        ! (s_log_store_env.store_status & APP_LOG_STORE_DUMP_BIT))
    {
        s_log_store_env.store_status |= APP_LOG_STORE_SAVE_BIT;
        if (s_log_store_ops.sem_give)
        {
            s_log_store_ops.sem_give();
        }
    }
    ...
}
```

app_log_store_save() caches logs into a ring buffer and adds a timestamp. When the data in the buffer reaches the waterline, the flag bit that is to be written into Flash will be set and the signal amount will be sent.

Note:

You can adjust the ring buffer size and waterline threshold according to project requirements, to save RAM space while avoiding buffer overflow. You can configure ring buffer size by using ring_buffer_init and adjust RAM space to store logs by modifying RAM_CODE_SPACE_SIZE in `SDK_Folder\platform\soc\linker\keil\flash_scatter_config.h`.

4.2 Log Scheduling API

Flash operations (including log writing, log export, and log clearing) are performed in app_log_store_schedule(). The Flash operation function that is registered during module initialization will be called when you perform Flash operations. The logic code for log storage and export is in *app_log_store.c*.

When logs are exported, the export success callback function s_log_dump_cbs->dump_process_cb will be called to transfer the exported data.

Path: `gr_libraries\app_log_store.c` under the project directory

Name: log_dump_from_flash()

```
static void log_dump_from_flash(void)
{
    ...
    if (s_log_store_ops.flash_read && need_dump_size)
    {
    ...
        if (s_log_dump_cbs->dump_process_cb)
        {
            s_log_dump_cbs->dump_process_cb(dump_buffer, dump_len);
        }

    }
    ...
}
```

During implementation of APP Log module, the data transmission API of Bluetooth LE Log Service is called in this callback function, to transmit the log data read from Flash from the device to the mobile phone through Bluetooth LE. The data transmission and peer command processing logics are implemented in *app_log_dump_port.c*, and Log Service is implemented in *lms.c*.

5 FAQ

This chapter describes possible problems, reasons, and solutions when you use APP Log module.

5.1 Why Are Logs Exported Through GRToolbox Missing?

- Description

  Logs exported through GRToolbox are missing.

- Analysis

  The ring buffer used to temporarily store logs overflows.

- Solution

  Increase the size of the ring buffer used to temporarily store logs. In an environment equipped with an operating system, you can try to increase the task priority of app_log_store_schedule().

5.2 Why Does Exporting of Historical Logs Through GRToolbox Fail?

- Description

  Only recent logs are exported through GRToolbox. Historical logs cannot be exported.

- Analysis

  RAM space for storing logs is insufficient, or logs are printed too frequently, thus the storage space overflows and overwrites historical logs.

- Solution

  - Increase the RAM space for log storage.
  - Delete unnecessary log print tasks.

