

# **GR5xx APP Log Application Note**

Version: 3.1

Release Date: 2023-08-08

#### Copyright © 2023 Shenzhen Goodix Technology Co., Ltd. All rights reserved.

Any excerption, backup, modification, translation, transmission or commercial use of this document or any portion of this document, in any form or by any means, without the prior written consent of Shenzhen Goodix Technology Co., Ltd. is prohibited.

#### **Trademarks and Permissions**

**GODIX** and other Goodix trademarks are trademarks of Shenzhen Goodix Technology Co., Ltd. All other trademarks and trade names mentioned in this document are the property of their respective holders.

#### Disclaimer

Information contained in this document is intended for your convenience only and is subject to change without prior notice. It is your responsibility to ensure its application complies with technical specifications.

Shenzhen Goodix Technology Co., Ltd. (hereafter referred to as "Goodix") makes no representation or guarantee for this information, express or implied, oral or written, statutory or otherwise, including but not limited to representation or guarantee for its application, quality, performance, merchantability or fitness for a particular purpose. Goodix shall assume no responsibility for this information and relevant consequences arising out of the use of such information.

Without written consent of Goodix, it is prohibited to use Goodix products as critical components in any life support system. Under the protection of Goodix intellectual property rights, no license may be transferred implicitly or by any other means.

#### 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:

- GR5xx user
- GR5xx developer
- GR5xx tester
- Hobbyist developer

#### **Release Notes**

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

#### **Revision History**

Version	Date	Description	
1.0	2022-05-10	Initial release	
3.0	2023-03-30	<ul> <li>Updated descriptions about GR5xx SoCs.</li> <li>Updated the code in sections "Log Output" and "Log Storage and Export".</li> </ul>	
3.1	2023-08-08	<ul> <li>Updated the file directory in "Adding Source Files".</li> <li>Updated the descriptions and code in "Log Output" and "Log Storage and Export".</li> </ul>	



# **Contents**

Preface	I
1 Introduction	1
2 Environment Setup	2
2.1 Preparation	2
3 Application of APP Log Module	3
3.1 Importing APP Log Module	3
3.1.1 Adding Source Files	3
3.1.2 Configuring Mode and Functionality	5
3.2 Module Initialization and Scheduling	6
3.2.1 Log Output	6
3.2.2 Log Storage and Export	8
3.3 Outputting Logs	11
3.4 Obtaining Logs	11
3.4.1 Obtaining Logs in Real Time	11
3.4.2 Exporting Stored Logs	14
4 Module Details	18
4.1 Log Transmission and Storage APIs	18
4.2 Log Scheduling API	19
5 FAQ	21
5.1 Why Are Logs Exported Through GRToolbox Missing?	21
5.2 Why Does Exporting of Historical Logs Through GRToolbox Fail?	



## 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 <a href="https://www.segger.com/downloads/">https://www.segger.com/downloads/</a> <a href="mailto:jlink/UM08001_JLink.pdf">jlink/UM08001_JLink.pdf</a> .
Keil User Guide	Offers detailed Keil operational instructions. Available at <a href="https://www.keil.com/support/">https://www.keil.com/support/</a> 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.
Reli MDK	Available at <a href="https://www.keil.com/download/product/">https://www.keil.com/download/product/</a> .
J-Link RTT Viewer (Windows)	A J-Link log output tool. Available at <a href="https://www.segger.com/products/debug-probes/j-link/tools/">https://www.segger.com/products/debug-probes/j-link/tools/</a>
J-Lilik KTT Viewei (Willdows)	rtt-viewer/.
GRUart (Windows)	A serial port debugging tool. Available in SDK_Folder\tools\GRUart.
GRToolbox (Android)	A Bluetooth LE debugging tool. Available in SDK_Folder\tools\GRToolbox.

### Note:

SDK\_Folder is the root directory of the GR5xx SDK in use.



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

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\_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 SD K\_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.



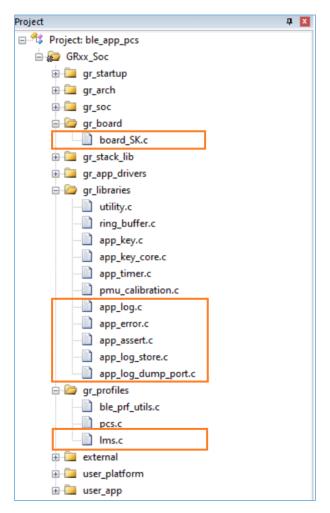


Figure 3-1 Adding source files into the project

(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:



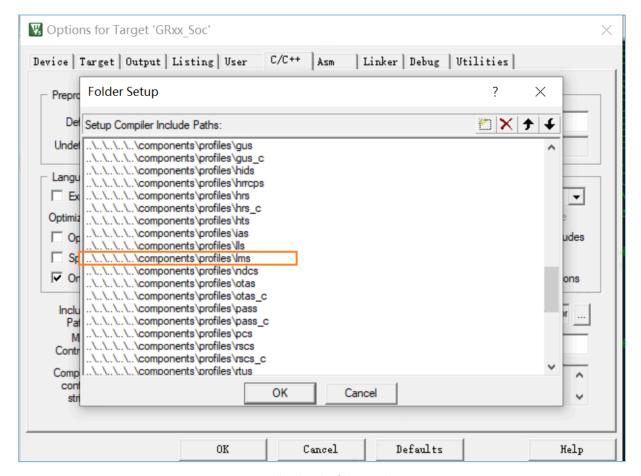


Figure 3-2 Adding header files into the project

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
```



Table 3-2 Macro description of APP Log module

Macro	Definition
	Enable/Disable APP Log module.
APP_LOG_ENABLE	0: Disable APP Log module.
	1: Enable APP Log module.
	Set the output mode of APP Log module.
APP LOG PORT	• 0: UART
	• 1: J-Link RTT
	• 2: ITM
	Enable/Disable the log storage functionality of APP Log module.
APP_LOG_STORE_ENABLE	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);
   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\_pe ripheral\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 + 0 \times 600000;
    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:



```
}
}
```

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\_pe ripheral\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.



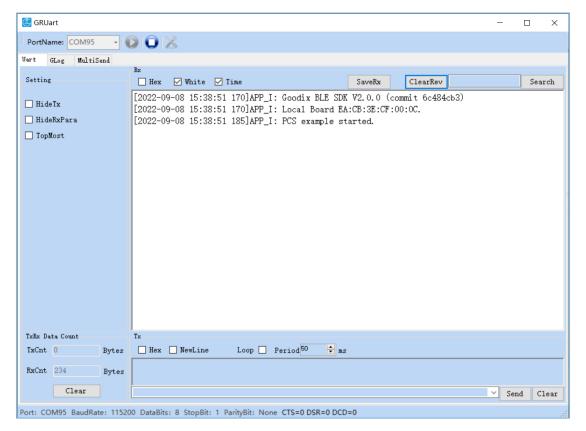


Figure 3-3 GRUart interface

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.



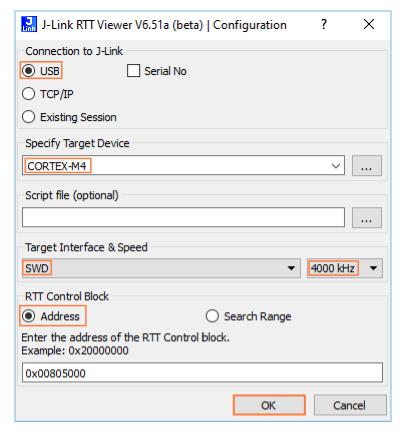


Figure 3-4 J-Link RTT Viewer configuration interface

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:



#### 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.

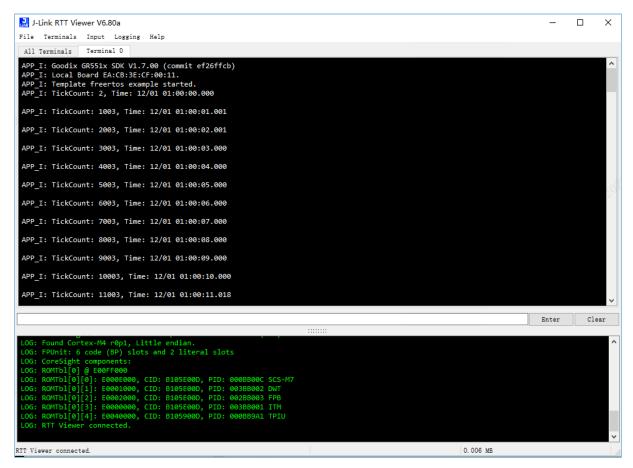


Figure 3-5 Log output interface of J-Link RTT Viewer

### 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.



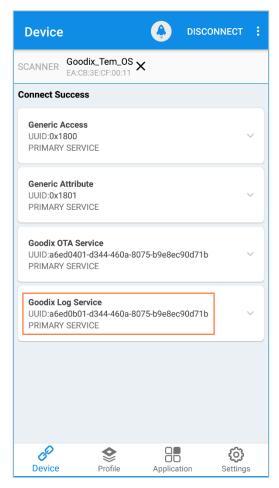


Figure 3-6 Successful discovery of GLS after connecting the phone to the Board through GRToolbox

### 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:



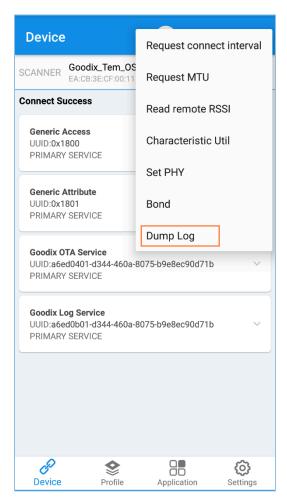


Figure 3-7 To output logs

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.

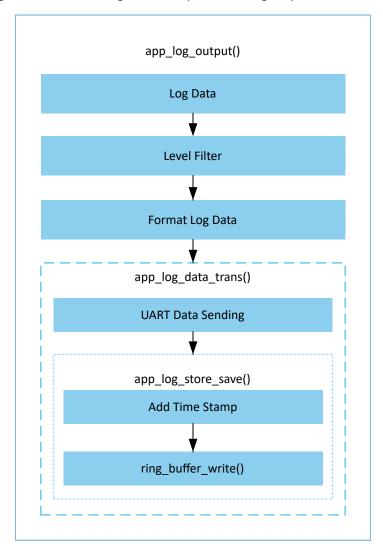


Figure 4-1 Calling relationship between log output functions



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();
        }
    }
    ...
}</pre>
```

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()

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.