

GR5xx Serial Port Profile Example Application

Version: 3.2

Release Date: 2025-02-17

Shenzhen Goodix Technology Co., Ltd.

Copyright © 2025 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

GOODIX 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: 26F, Goodix Headquarters, No.1 Meikang Road, Futian District, Shenzhen, China

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

Website: www.goodix.com

Preface

Purpose

This document introduces how to use and verify the Serial Port Profile (SPP) example in the Bluetooth Low Energy (Bluetooth LE) GR5xx Software Development Kit (SDK), to help users quickly get started with secondary development.

Audience

This document is intended for:

- Device user
- Developer
- Test engineer
- Hobbyist developer
- Technical writer

Release Notes

This document is the fourth release of *GR5xx Serial Port Profile Example Application*, corresponding to Bluetooth LE GR5xx System-on-Chip (SoC) series.

Revision History

Version	Date	Description
1.0	2023-01-10	Initial release
3.0	2023-03-30	Updated descriptions about GR5xx SoCs.
3.1	2023-11-06	Updated the approaches for obtaining GProgrammer, GRToolbox, and GRUart.
3.2	2025-02-17	Updated the code in "Receiving Data and Transmitting the Data to Serial Ports".

Contents

Preface	I
1 Introduction	1
2 Profile Overview	2
3 Initial Operation	4
3.1 Preparation	4
3.2 Firmware Programming	4
3.3 Test and Verification	5
4 Application Details	9
4.1 Running Procedures	9
4.2 Major Code	9
4.2.1 Enabling Notification of Data TX & Data Flow Control Characteristics	9
4.2.2 Receiving Data and Transmitting the Data to Serial Ports	10
4.2.3 Receiving Data from Serial Ports and Transmitting the Data to Initiator	10
5 FAQ	13
5.1 Why Is the Data Split into Smaller Packets and Transmitted?	13
5.2 Why Is the Data Sent in String but Received in Hexadecimal?	14
6 Appendix: Throughput Test Result	17

1 Introduction

Serial Port Profile (SPP) defines how to pass through data from virtual serial ports to peer Bluetooth Low Energy (Bluetooth LE) devices by adopting Bluetooth LE technology.

Bluetooth Special Interest Group (Bluetooth SIG) does not define standard profiles for Bluetooth LE serial port passthrough. Therefore, to make Goodix-customized SPP user-friendly, this document introduces how to use and verify the Goodix SPP example in the GR5xx Software Development Kit (SDK).

Before getting started, you can refer to the documents listed below.

Name	Description
GR5xx Sample Service Application and	Introduces how to apply and customize Goodix Sample Service in developing Bluetooth LE
Customization	applications based on GR5xx SDK.
Developer guide of the specific GR5xx SoC	Introduces GR5xx SDK and how to develop and debug applications based on the SDK.
Bluetooth Core Spec	Offers official Bluetooth standards and core specification from Bluetooth SIG.
Bluetooth GATT Spec	Provides details about Bluetooth profiles and services. Available at https://www.bluetooth.com/specifications/gatt .
J-Link/J-Trace User Guide	Provides J-Link operational instructions. Available at <u>https://www.segger.com/downloads/</u> jlink/UM08001_JLink.pdf.
Keil User Guide	Offers detailed Keil operational instructions. Available at <u>https://www.keil.com/support/</u> man/docs/uv4/.

Table 1-1 Reference documents

2 Profile Overview

Goodix SPP defines two device roles:

- Initiator: the device that issues a connection request to another device
- Acceptor: the device that waits for connection requests from other devices

The figure below shows how the two kinds of devices get connected and pass through data.



Figure 2-1 Acceptor-Initiator interaction process

Goodix SPP only defines the data pass-through service of GR5xx System-on-Chips (SoCs) (Goodix UART Service, GUS). The service is customized by Goodix, with the 128-bit vendor-specific UUID of A6ED0201-D344-460A-8075-B9E8EC90D71B to transmit data, and to update Bluetooth LE data flow control.

GUS is characterized by:

- RX characteristic: Receives the written data from the initiator.
- TX characteristic: Sends data through serial ports to the initiator.
- Flow Control characteristic: Updates the capacities of the acceptor and the initiator in receiving Bluetooth LE data (0x00: Cannot receive more Bluetooth LE data; 0x01: Can receive more Bluetooth LE data).

These characteristics are described in detail as follows:

Table 2-1 GUS characteristics

Characteristic	UUID	Туре	Support	Security	Property
RX	A6ED0202-D344-460A-8075-B9E8EC90D71B	128 bits	Mandatory	None	Write
ТХ	A6ED0203-D344-460A-8075-B9E8EC90D71B	128 bits	Mandatory	None	Notify
Flow Control	A6ED0204-D344-460A-8075-B9E8EC90D71B	128 bits	Mandatory	None	Notify and Write

3 Initial Operation

This chapter introduces how to quickly verify an SPP example in the GR5xx SDK.

Note:

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

3.1 Preparation

Perform the following tasks before using and modifying the Goodix SPP example.

• Hardware preparation

Table 3-1 Hardware preparation

Name Description	
J-Link debug probe	JTAG emulator launched by SEGGER. For more information, visit https://www.segger.com/products/
	debug-probes/j-link/
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)

• Software preparation

Table 3-2 Software preparation

Name	Description
Windows	Windows 7/Windows 10
J-Link driver	A J-Link driver. Available at <u>https://www.segger.com/downloads/jlink/</u> .
Keil MDK5	An integrated development environment (IDE). MDK-ARM 5.20 or later is required. Available at
	https://www.keil.com/download/product/.
GRToolbox (Android)	A Bluetooth LE debugging tool. Available at <u>https://www.goodix.com/en/software_tool/grtoolbox</u>
GRUart (Windows)	A serial port debugging tool. Available at <u>https://www.goodix.com/en/download?</u>
GROart (Windows)	objectId=43&objectType=software.
GProgrammer (Windows)	A programming tool. Available at <u>https://www.goodix.com/en/software_tool/gprogrammer_ble</u> .

3.2 Firmware Programming

The source code of the Goodix SPP example is in SDK_Folder\projects\ble\ble_peripheral\ble_app_ uart.

You can download *ble_app_uart.bin* to an SK Board through GProgrammer. For details, see *GProgrammer User Manual*.

🛄 Note:

ble_app_uart.bin is in SDK_Folder\projects\ble\ble_peripheral\ble_app_uart\build.

3.3 Test and Verification

When an SK Board, GRToolbox, and GRUart are ready, test and verify the SPP example. Steps are described as follows:

1. Connect to the SK Board through GRToolbox.

Launch GRToolbox on an Android mobile phone to search for the device **Goodix_UART** (advertising name, which can be modified in the *user_app.c* file).



Figure 3-1 Discovering **Goodix_UART** on the mobile phone

🛄 Note:

Screenshots of GRToolbox in this document are for reference only, to help users better understand the software operation. In the case of interface differences due to version changes, the interface of GRToolbox in practice shall prevail.

Tap **CONNECT** to connect to **Goodix_UART**, and the screen shows **Goodix UART Service** information, including **TX Characteristic**, **RX Characteristic**, and **Flow Control Characteristic**, as shown in the figure below.



Figure 3-2 Discovering Goodix UART Service on the mobile phone

2. Send data through GRToolbox.

Enable notifications for **TX Characteristic** and **Flow Control Characteristic** in GUS on the peer device through GRToolbox. The mobile phone displays as below:

Device	÷	DISCO	NNECT :
SCANNER	Goodix_UART EA:CB:3E:CF:00:13	×	
Connect S	uccess		
UUID:a6e	JART Service ed0201-d344-460a Y SERVICE	8075-b9e8ec90d71b	^
Tx Cha	racteristic		8
	6ed0202-d344-460 ties:NOTIFY	a-8075-b9e8ec90d71b	
Desc	criptors:		
UUID	nt Characteristic Co D:0x2902 Ie:Notification is er		R
Rx Cha	racteristic		W
1000000	6ed0203-d344-460 ties:WRITE, WRITE	a-8075-b9e8ec90d71b NO RESPONSE	
	ontrol Characterist	tic	0 🛛
	ties:WRITE, NOTIF		
	criptors:		
Clier	nt Characteristic Co	onfiguration	R
100000	0:0x2902 e:Notification is er	nabled	
R	\$		ത
Device	Profile	Application	Settings

Figure 3-3 Interface after enabling notifications for TX Characteristic and Flow Control Characteristic

Write data (such as 12345678) to GUS and tap SEND.

Device	DISCONNECT
Connect Success	
Goodix UART Service UUID:a6ed0201-d344-460a-8075-b PRIMARY SERVICE	De8ec90d71b
Write Data	
Data: 12345678	0
Data Format: String 🔹	
Write Method: Write Comma	and 🝷
SAVE	CANCEL SEND
Properties:wkite, wkite NO KES	PUNSE
Flow Control Characteristic UUID:a6ed0204-d344-460a-8075- Properties:WRITE, NOTIFY Descriptors:	🦁 📎 b9e8ec90d71b
Client Characteristic Configura UUID:0x2902 Value:Notification is enabled	tion R
Device Profile A	pplication Settings

Figure 3-4 Entering **RX Characteristic** attributes

Data sent through GRToolbox is shown in the **Receive Data** area of GRUart, as shown in the figure below.

PortName: COM29 Image: Comparison of the state of the stateo
Rx Setting Hex White Time SaveRx ClearRev Search HideTx APP_I: Goodix BLE SDK HideKxFara APP_I: Local Board EA:CB:3E:CF:00:13. HideKxFara APP_I: Goodix UART example started. TopMost APP_I: Connected with the peer 55:DC:5F:E3:09:45.
Setting Hex ✓ White Time SaveRx ClearRev Search APP_I: Goodix BLE SDK HideTx APP_I: Local Board EA:CB:3E:CF:00:13. HideRxPara APP_I: Goodix UART example started. TopMost APP_I: Connected with the peer 55:DC:5F:E3:09:45.
HideTx APP_I: Goodix BLE SDK HideRxPara APP_I: Local Board EA:CB:3E:CF:00:13. HideRxPara APP_I: Goodix UART example started. TopMost APP_I: Connected with the peer 55:DC:5F:E3:09:45.
Txfx Data Count Tx
TxCnt 0 Bytes Hex 🗸 NewLine Loop 🗖 Period ⁵⁰ ≑ ms
RxCnt 183 Bytes
Clear Send Clear

Figure 3-5 Printing data sent from GRToolbox on GRUart

3. Send data through GRUart.

Enter **abcdefgh** in the **Send data** pane in GRUart, and click **Send**.

The Value of TX Characteristic in GRToolbox shows the data sent from GRUart, as shown in the figure below.



Figure 3-6 Showing data sent from GRUart on GRToolbox

If the two applications function as described above, the Goodix SPP example runs successfully.

4 Application Details

This chapter introduces the running procedures and major code of the Goodix SPP example.

4.1 Running Procedures

After the initiator discovers and connects to the Goodix SPP example, the main process is shown in the figure below:



Figure 4-1 Running procedures of the Goodix SPP example

4.2 Major Code

The sections below introduce major code related to interactions between the initiator and the acceptor.

4.2.1 Enabling Notification of Data TX & Data Flow Control Characteristics

After the initiator sends a command about enabling the notification of GUS **TX Characteristic** on the acceptor, GUS parses the command and reports the "GUS_EVT_TX_PORT_OPENED" event to the application layer to enable the notification of **TX Characteristic**. This enables the acceptor to transmit data from serial ports to the initiator.

After the initiator sends a command about enabling the notification of GUS Flow Control Characteristic on the acceptor, GUS parses the command and reports the "GUS_EVT_FLOW_CTRL_ENABLE" event to the application layer to enable the notification of Flow Control Characteristic. This enables the acceptor to notify the initiator of the capacity for receiving Bluetooth LE data.

Path: user_app\user_app.c under the project directory

```
Name: gus_service_process_event();
```

```
static void gus_service_process_event(gus_evt_t *p_evt)
```

```
switch (p_evt->evt_type)
{
    case GUS_EVT_TX_PORT_OPENED:
        transport_flag_set(GUS_TX_NTF_ENABLE, true);
        break;
    case GUS_EVT_FLOW_CTRL_ENABLE:
        transport_flag_set(BLE_FLOW_CTRL_ENABLE, true);
        break;
    ...
    }
}
```

4.2.2 Receiving Data and Transmitting the Data to Serial Ports

After the acceptor receives Bluetooth LE data from the initiator, GUS reports the "GUS_EVT_RX_DATA_RECEIVED" event to the application layer, which calls ble_to_uart_push() to store the data in the corresponding ring buffers.

Path: user_app\user_app.c under the project directory

```
Name: gus_service_process_event();
```

```
static void gus_service_process_event(gus_evt_t *p_evt)
{
    switch (p_evt->evt_type)
    {
        ...
        case GUS_EVT_RX_DATA_RECEIVED:
            ble_to_uart_push(p_evt->p_data, p_evt->length);
            break;
        ...
    }
}
```

The function transport_schedule() runs in the while loop of the main() function and polls the ring buffers. When new data in the ring buffers is detected, call the transport_uart_data_send() function. The function retrieves data from the ring buffers and transmits the data to serial ports.

Path: user_app\transport_scheduler.c under the project directory

Name: transport_uart_data_send()

4.2.3 Receiving Data from Serial Ports and Transmitting the Data to Initiator

After receiving data from the serial ports, the acceptor keeps the data received from serial port events in the ring buffers temporarily supported by the uart_evt_handler() function.

Path: user_platform\user_periph_setup.c under the project directory

Name: uart_evt_handler();

```
static void uart_evt_handler(app_uart_evt_t *p_evt)
{
    if (APP_UART_EVT_RX_DATA == p_evt->type)
    {
        uart_to_ble_push(s_uart_rx_buffer, p_evt->data.size);
        app_uart_dma_receive_async(APP_UART_ID, s_uart_rx_buffer, UART_RX_BUFFER_SIZE);
    }
    else if (APP_UART_EVT_TX_CPLT == p_evt->type)
    {
        update_ble_flow_ctrl_state();
    }
}
```

When there are no Bluetooth LE data transmission tasks, the function transport_schedule() calls the function transport_ble_data_send() to poll the ring buffers. If there is data to be transmitted in the ring buffers, the Bluetooth LE data transmission tasks are executed.

Path: user_app\transport_scheduler.c under the project directory

Name: transport_ble_data_send();

When one Bluetooth LE data transmission task is completed, GUS reports the "GUS_EVT_TX_DATA_SENT" event to the application layer which calls the function transport_ble_continue_send() to check the ring buffers. If there is data to be transmitted in the ring buffers, the Goodix SPP continues retrieving the data and transmitting the data to the initiator.

Path: user_app\transport_scheduler.c under the project directory

Name: transport_ble_continue_send();

```
void transport_ble_continue_send(void)
{
    ...
    transport_flag_set(BLE_SCHEDULE_ON, true);
    // Read data from m uart rx ring buffer and send to peer via BLE.
```

5 FAQ

This chapter introduces possible problems, reasons, and solutions during verification and application of the Goodix SPP example.

5.1 Why Is the Data Split into Smaller Packets and Transmitted?

Description

When the data input through GRUart is more than 20 bytes, the data is split into smaller packets and transmitted in several times.

Analysis

Before the initiator and the acceptor exchange the maximum transmission unit (MTU), the MTU size is 23 bytes by default, including the 1-byte opcode, and the 2-byte attribute handle. Therefore, the length for one data transmission is limited to 20 bytes.

When the length of data to be transmitted exceeds 20 bytes, the data is transmitted in sequence and in units of no more than 20 bytes in several times.

This problem can be solved by modifying the MTU value.

Solution

Tap **•** > **Request MTU** in the upper-right corner of GRToolbox, as shown in the figure below.

Device		Request connect interv	al
SCANNER	Goodix_UART EA:CB:3E:CF:00:13	Request MTU	
Connect S	uccess	Read remote RSSI	
UUID:a6e	ART Service ed0201-d344-460a-8 7 SERVICE	Bond 8075-b9e8ec90d71b	^
UUID:a	racteristic 6ed0202-d344-460a ties:NOTIFY	-8075-b9e8ec90d71b	
	riptors:		
UUIC	nt Characteristic Co 0:0x2902 e:Notification is ena		
Rx Cha	racteristic	W	
	6ed0203-d344-460a ties:WRITE, WRITE N	-8075-b9e8ec90d71b NO RESPONSE	
Flow C	ontrol Characteristi	c 🛛 🕐 🚫	
Propert	ties:WRITE, NOTIFY	-8075-b9e8ec90d71b	
	criptors:	•	
UUIC	nt Characteristic Co 0:0x2902 e:Notification is ena		
B	٢	8 0	
Device	Profile	Application Setting	S

Figure 5-1 Choosing Request MTU

Enter a customized MTU value, such as **400** bytes, and tap **OK** to update the value (range of MTU value: 23 bytes to 512 bytes).



Figure 5-2 Setting the MTU value

5.2 Why Is the Data Sent in String but Received in Hexadecimal?

Description

The data sent through serial ports is in strings (such as "abcdefgh"), but the received data in GRToolbox is in hexadecimal (unit: byte).

Analysis

The data format is incorrect in settings.

Solution

The data format in GRToolbox (both for received data and transmitted data) can be set in string or in byte. As shown in the figure below, the received data is presented in byte.



Figure 5-3 Format of received data in GRToolbox (in byte)

Tap Value and the data format menu pops up.

	NER Goodix_UART ×		
Conn	ect Success		
1	Select data format	_	
	HEX		
	ASCII		
L	UTF-8		
	UNICODE		
	GB2312		
	veaunpluia.		
	Client Characteristic Configuration UUID:0x2902 Value:Notification is enabled	ß	
De	Profile Applicat	ion Settings	

Figure 5-4 Choosing a data format

Choose ASCII and tap Confirm, and the string "abcdefgh" is presented, as shown in the figure below.



Figure 5-5 Presenting the string "abcdefgh"

6 Appendix: Throughput Test Result

Bluetooth LE throughput tests on Goodix SPP are performed based on an SK Board.

The test results include baud rates for serial ports (115200 bps, 230400 bps, and 460800 bps), and the throughputs in cases of 1 M PHY and 2 M PHY, and in different pass-throughput modes.

Baud Rate (bps)	Pass-through Mode	1M PHY	2М РНҮ
	accepter \rightarrow initiator	10.032 KB/s	10.246 KB/s
115200	accepter \leftarrow initiator	10.015 KB/s	10.167 KB/s
	accepter \leftrightarrow initiator	19.534 KB/s	19.758 KB/s
	accepter \rightarrow initiator	20.329 KB/s	21.011 KB/s
230400	accepter \leftarrow initiator	20.009 KB/s	19.907 KB/s
	accepter \leftrightarrow initiator	40.069 KB/s	41.01 KB/s
	accepter \rightarrow initiator	37.38 KB/s	37.826 KB/s
460800	accepter \leftarrow initiator	37.38 KB/s	37.648 KB/s
	accepter \leftrightarrow initiator	70.243 KB/s	71.141 KB/s

Table 6-1 Throughput in dif	fferent modes
-----------------------------	---------------