GR551x Serial Port Profile Example Application

Version: 1.7

Release Date: 2020-12-15

Shenzhen Goodix Technology Co., Ltd.

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: 2F. & 13F., Tower B, Tengfei Industrial Building, Futian Free Trade Zone, Shenzhen, China

TEL: +86-755-33338828 FAX: +86-755-33338099

Website: www.goodix.com

Preface

Purpose

This document introduces how to use and verify the Serial Port Profile (SPP) example in a GR551x SDK, to help users quickly get started with secondary development.

Audience

This document is intended for:

- GR551x user
- GR551x developer
- GR551x tester
- Hobbyist developer
- Technical writer

Release Notes

This document is the fifth release of *GR551x Serial Port Profile Example Application*, corresponding to GR551x SoC series.

Revision History

Version	Date	Description
1.0	2019-12-08	Initial release
1.3	2020-03-16	Updated the release time in the footers.
1.5	2020-05-30	Updated two parameters (from GUS_NTF_ENABLE and BLE_TX_CPLT to GUS_TX_NTF_ENABLE) in “Chapter 4 Application Details”.
1.6	2020-06-30	Updated the document version based on SDK changes.
1.7	2020-12-15	Updated GRToolbox UI figures based on software update.

Contents

Preface I

1 Introduction 1

2 Profile Overview 2

3 Initial Operation 4

3.1 Preparation 4

3.2 Hardware Connection 4

3.3 Firmware Download 5

3.4 Serial Port Settings 5

3.5 Test and Verification 6

4 Application Details 9

4.1 Project Directory 9

4.2 Main Process and Code 9

5 FAQ 14

5.1 Why does the Mobile Phone Receive Data in Multiple Units, with Each Less Than or Equal to 20 Bytes? 14

5.2 Why Is the Data Sent Through Serial Ports in String, But the Data Received in Hexadecimal? 15

6 Appendix: Throughput Test Result 18

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 pass-through. Therefore, to make Goodix-customized SPP user-friendly, this document introduces how to use and verify the Goodix SPP example in a GR551x SDK.

Before you get started, it is recommended to refer to the documents listed below.

Table 1-1 Reference documents

Name	Description
GR551x Sample Service Application and Customization	Introduces how to apply and customize Goodix Sample Service in developing Bluetooth LE applications based on GR551x SDK.
GR551x Developer Guide	Provides GR551x software and hardware introduction, quick start, and resource overview.
Bluetooth Core Spec v5.1	Offers official Bluetooth standards and core specification (v5.1) from Bluetooth SIG. Available at https://www.bluetooth.com/specifications/bluetooth-core-specification/.
Bluetooth GATT Spec	Provides details about Bluetooth profiles and services. Available at www.bluetooth.com/specifications/gatt.
J-Link/J-Trace User Guide	Provides J-Link operational instructions. Available at www.segger.com/downloads/jlink/UM08001_JLink.pdf.
Keil User Guide	Offers detailed Keil operational instructions. Available at www.keil.com/support/man/docs/uv4/.

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.

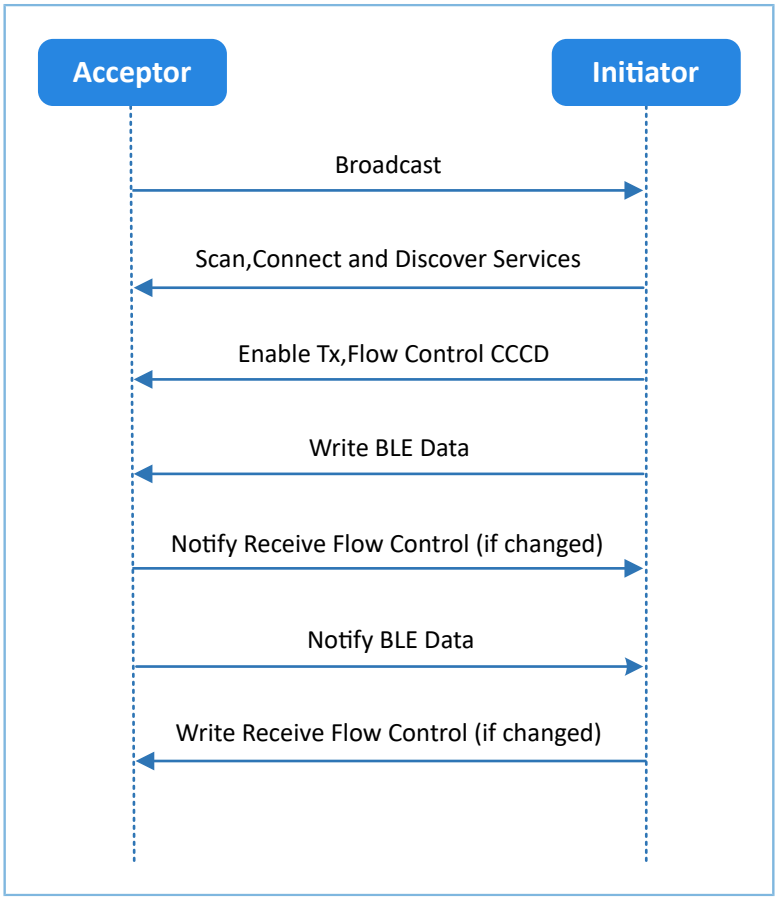

Figure 2-1 Acceptor-Initiator interaction process

Goodix SPP only defines the data pass-through service of GR551x (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	Type	Support	Security	Property
RX	A6ED0202-D344-460A-8075-B9E8EC90D71B	128 bits	Mandatory	None	Write
TX	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 a GR551x SDK.

Note:

SDK_Folder is the root directory of GR551x SDK.

3.1 Preparation

Get ready as described below, before verifying and testing the Goodix SPP example.

- **Hardware preparation**

Table 3-1 Hardware preparation

Item	Description
J-Link debug probe	JTAG emulator launched by SEGGER. For more information, visit www.segger.com/products/debug-probes/j-link/.
Development board	GR5515 Starter Kit Board (GR5515 SK Board)
Connection Cable	Micro USB 2.0 cable

- **Software preparation**

Table 3-2 Software preparation

Item	Description
Windows	Windows 7/Windows 10
J-Link driver	A J-Link driver. Available at www.segger.com/downloads/jlink/.
Keil MDK5	An integrated development environment (IDE). Available at www.keil.com/download/product/.
GRToolbox (Android)	A GR551x Bluetooth LE debugging tool. Available in `SDK_Folder\tools\GRToolbox`.
GRUart (Windows)	A GR551x serial port debugging tool. Available in `SDK_Folder\tools\GRUart`.
GProgrammer (Windows)	A GR551x programming tool. Available in `SDK_Folder\tools\GProgrammer`.

3.2 Hardware Connection

Connect a GR5515 SK Board to a PC with a Micro USB 2.0 cable, as shown in the figure below.

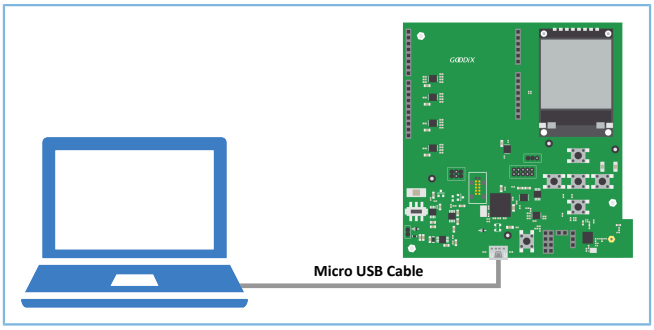

Figure 3-1 Hardware connection

3.3 Firmware Download

For instructions on how to download *ble_app_uart_fw.bin* firmware to the board, see *GProgrammer User Manual*.

Note:

ble_app_uart_fw.bin is in SDK_Folder\projects\ble\ble_peripheral\ble_app_uart\build.
SDK_Folder is the root directory of GR551x SDK.

3.4 Serial Port Settings

Start GRUart, and configure the serial ports according to the parameters in the table below.

Table 3-3 Configuring serial port parameters on GRUart

PortName	BaudRate	DataBits	Parity	StopBits	Flow Control
Select on demand	115200	8	None	1	Uncheck

When configuration is complete, click **Open Port**, as shown in the figure below.

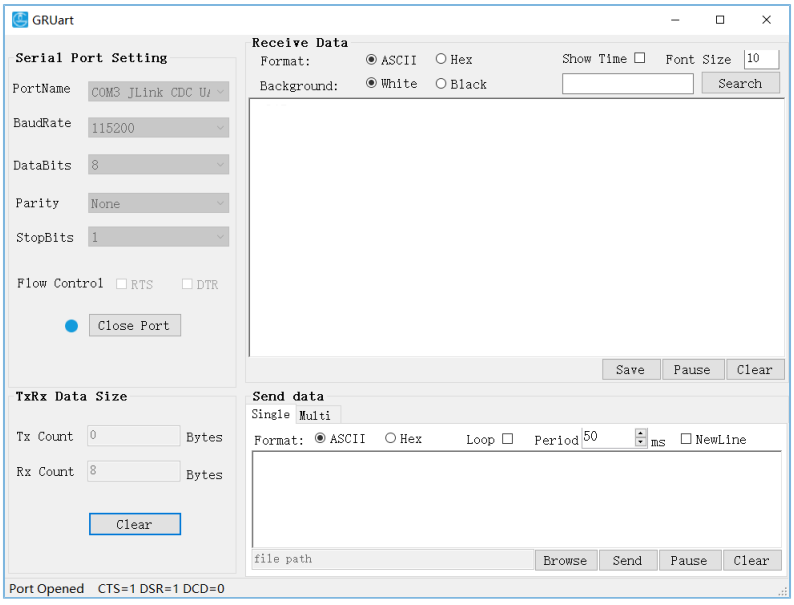

Figure 3-2 Serial port settings on GRUart

3.5 Test and Verification

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).

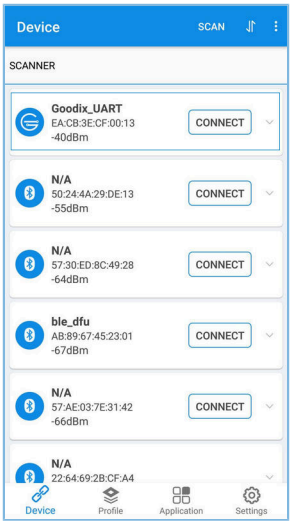

Figure 3-3 Discovering **Goodix_UART** on the mobile phone

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.

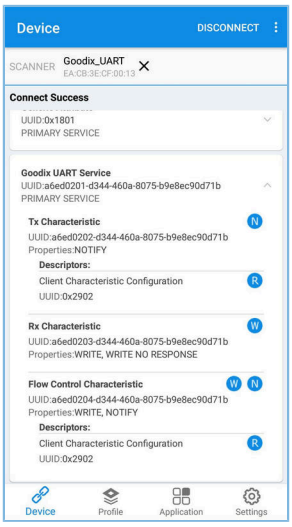

Figure 3-4 Discovering **Goodix UART Service** on the mobile phone

After getting prepared as mentioned above, follow the steps below to verify the Goodix SPP example.

1. 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:

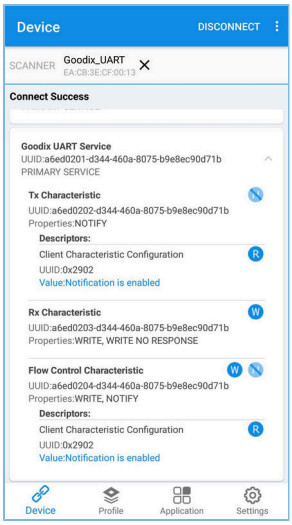

Figure 3-5 Interface after enabling notifications for **TX Characteristic** and **Flow Control Characteristic**

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

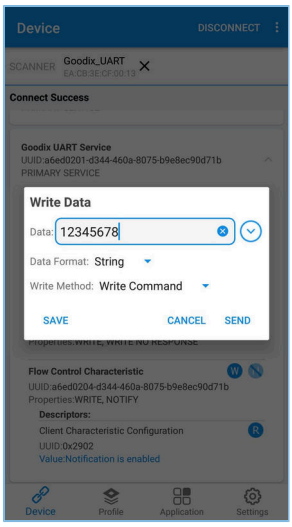

Figure 3-6 Entering **RX Characteristic** attributes

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

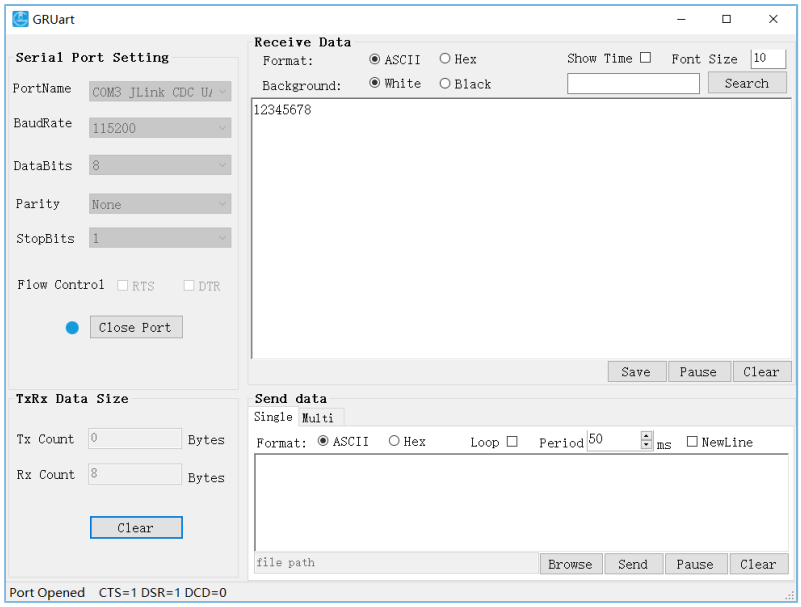

Figure 3-7 Printing data sent from GRToolbox on GRUart

2. 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.

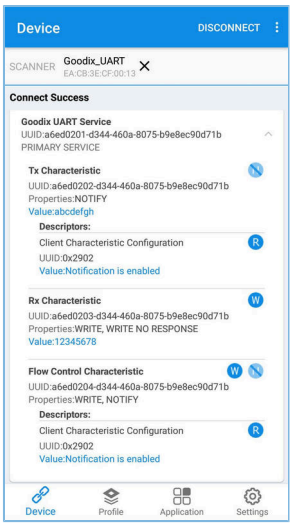

Figure 3-8 Showing data sent from GRUart on GRToolbox

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

4 Application Details

This chapter introduces the project directory, main processes, and some critical code of the Goodix SPP example.

4.1 Project Directory

The source code and the project file of the Goodix SPP example are in SDK_Folder\projects\ble\ble_peripheral\ble_app_uart, and project file is in the folder Keil_5.

Double-click the project file *ble_app_uart.uvprojx* to check the structure of the project directory of the Goodix SPP example, ble_app_uart. Details of related files are described in the table below.

Table 4-1 File descriptions of ble_app_uart

Group	File	Description
gr_libraries	ring_buffer.c	This file implements operations with ring buffers.
gr_profiles	ble_prf_utils.c	This file contains profile-related operational tools.
	gus.c	This file implements Goodix UART Service.
user_callback	user_gap_callback.c	This file implements GAP Callback, such as connection, disconnection, and GAP parameter update.
	user_gatt_common_callback.c	This file implements GATT Common Callback, such as MTU exchange.
user_platform	user_periph_setup.c	This file initializes UART, device address, and sets power management modes.
user_app	main.c	This file contains main() function.
	user_app.c	This file sets GUS advertising parameters.
	transport_scheduler.c	This file implements the distribution of serial port data and Bluetooth LE data.

4.2 Main Process and Code

After powering on the GR5515 SK Board downloaded with the firmware of Goodix SPP example, the SK Board automatically initializes the peripherals, BLE Protocol Stack, and GUS.

Note:

Locations of the main logic code of the Goodix SPP example:

- *user_app.c* and *transport_scheduler.c* in user_app in the Keil project directory tree
- *user_periph_setup.c* in user_platform in the Keil project directory tree

---

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

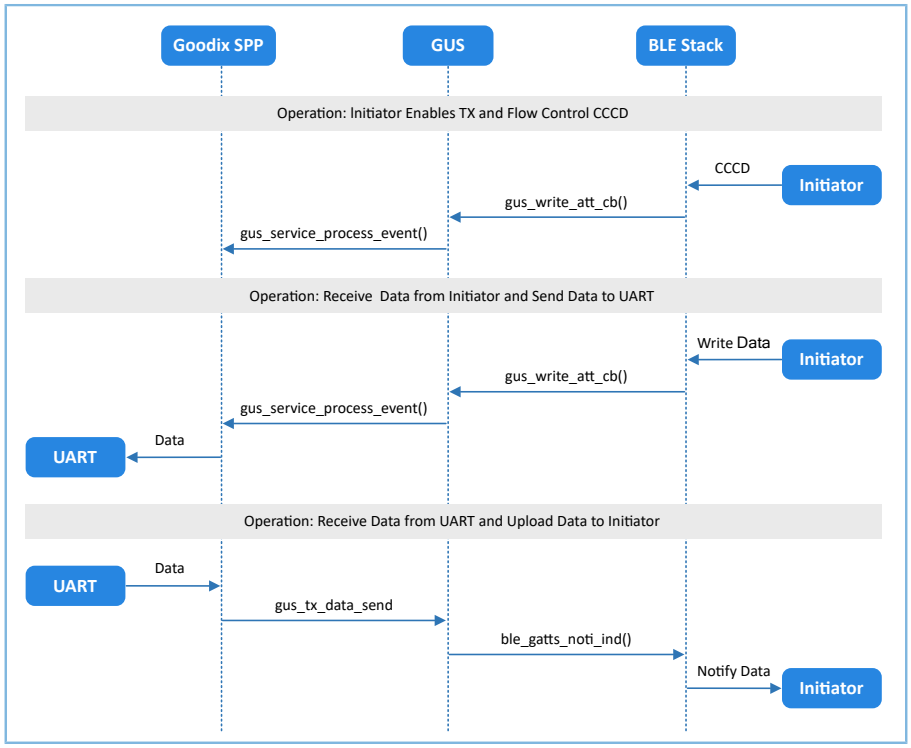

Figure 4-1 Main process of the Goodix SPP example

1. The initiator enables or disables Bluetooth LE data transmission and Bluetooth LE data flow control.

   This process is mainly implemented by the gus_service_process_event() function in *user_app.c*.

   - When notification of GUS **TX Characteristic** on the acceptor is enabled by the initiator, the Goodix SPP receives GUS_EVT_TX_PORT_OPENED. This enables the acceptor to transmit data from serial ports to the initiator.
   - When notification of GUS Flow Control Characteristic on the acceptor is enabled by the initiator, the Goodix SPP receives GUS_EVT_FLOW_CTRL_ENABLE. This enables the acceptor to notify the initiator of the capacity for receiving Bluetooth LE data.
   - When the initiator disables notifications of the above two characteristics, Goodix SPP sets corresponding labels as False.

   A code snippet is as follows:

```
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_TX_PORT_CLOSED:
            transport_flag_set(GUS_TX_NTF_ENABLE, false);
            break;
        ...
        case GUS_EVT_FLOW_CTRL_ENABLE:
```

```
            transport_flag_set(BLE_FLOW_CTRL_ENABLE, true);
            break;

        case GUS_EVT_FLOW_CTRL_DISABLE:
            transport_flag_set(BLE_FLOW_CTRL_ENABLE, false);
            break;
        ...
    }
}
```

2. The acceptor receives data from the initiator and transmits the data to serial ports.

   After receiving Bluetooth LE data from the initiator, the Goodix SPP receives GUS_EVT_RX_DATA_RECEIVED, and stores the data in the corresponding ring buffers. Poll the tasks in the ring buffers by executing the function transport_schedule(). 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. A code snippet is as follows:

```
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;
        ...
    }
}

void transport_schedule(void)
{
    transport_uart_data_send();
    ...
}

static void transport_uart_data_send(void)
{
    uint16_t read_len;
    uint16_t items_avail;

    items_avail = ring_buffer_items_count_get(&s_ble_RX_ring_buffer);

    if (items_avail > 0)
    {
        read_len = ring_buffer_read(&s_ble_RX_ring_buffer, s_uart_TX_data,
                                    UART_ONCE_SEND_SIZE);
        transport_flag_set(UART_TX_CPLT, false);
        uart_TX_data_send(s_uart_TX_data, read_len);
    }
}
```

3. The acceptor receives data from the serial ports and transmits the data to the 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 app_uart_evt_handler() function. A code snippet is as follows:

```
static void app_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_receive_async(APP_UART_ID, s_uart_RX_buffer,
                               UART_RX_BUFFER_SIZE);
    }
    ...
}
```

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. A code snippet is as follows:

```
void transport_schedule(void)
{
    ...
    if (transport_flag_cfm(GUS_TX_NTF_ENABLE) &&
        transport_flag_cfm(BLE_TX_CPLT) &&
        transport_flag_cfm(BLE_TX_FLOW_ON) &&
        transport_flag_cfm(BLE_SCHEDULE_ON))
    {
        transport_ble_data_send();
    }
}

static void transport_ble_data_send(void)
{
    uint16_t read_len;
    uint16_t items_avail;

    items_avail = ring_buffer_items_count_get(&s_uart_RX_ring_buffer);

    if (items_avail > 0)
    {
        read_len = ring_buffer_read(&s_uart_RX_ring_buffer, s_ble_TX_data,
                                    s_mtu_size - 3);
        transport_flag_set(BLE_TX_CPLT, false);
        gus_TX_data_send(0, s_ble_TX_data, read_len);
    }
}
```

When one Bluetooth LE data transmission task is completed, the Goodix SPP receives GUS_EVT_TX_DATA_SENT, and 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.

```
void transport_ble_continue_send(void)
{
    uint16_t read_len;
    uint16_t items_avail;

    transport_flag_set(BLE_SCHEDULE_ON, true);

    // Read data from m_uart_RX_ring_buffer and send to peer via BLE.
    if (transport_flag_cfm(BLE_TX_FLOW_ON))
    {
        items_avail = ring_buffer_items_count_get(&s_uart_RX_ring_buffer);

        if (items_avail > 0)
```

```
        {
            read_len = ring_buffer_read(&s_uart_RX_ring_buffer, s_ble_TX_data,
                                        s_mtu_size - 3);
            transport_flag_set(BLE_TX_CPLT, false);
            transport_flag_set(BLE_SCHEDULE_ON, false);
            gus_TX_data_send(0, s_ble_TX_data, read_len);
        }
    }
}
```

5 FAQ

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

5.1 Why does the Mobile Phone Receive Data in Multiple Units, with Each Less Than or Equal to 20 Bytes?

- 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.

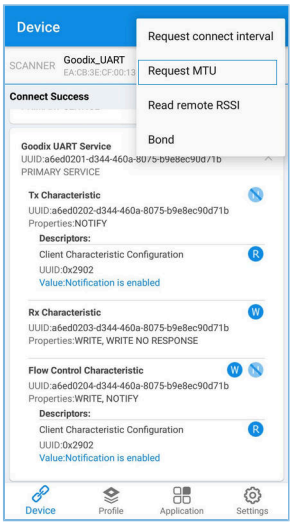

Figure 5-1 Choosing **Request MTU**

Note:

MTU can only be updated once in one connection. If an MTU update fails, it is probably because one MTU update has been made before.

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).

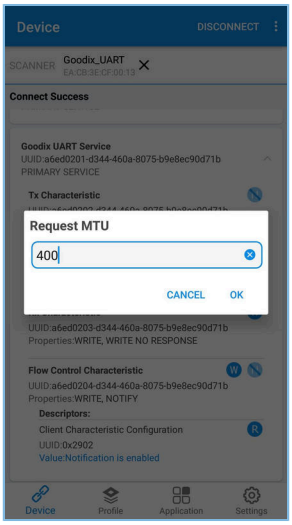

Figure 5-2 Setting the MTU value

5.2 Why Is the Data Sent Through Serial Ports in String, But the Data 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.

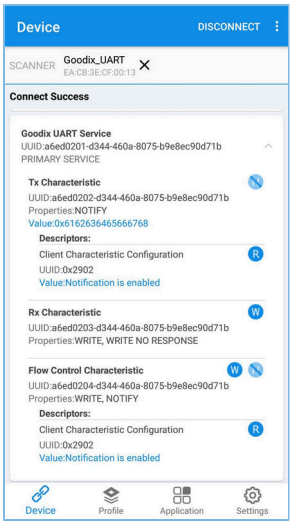

Figure 5-3 Format of received data in GRToolbox (in byte)

Tap **Value** and the data format menu pops up.

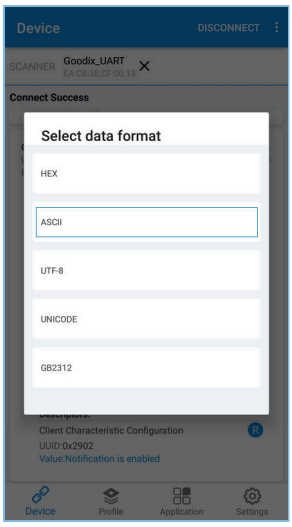

Figure 5-4 Choosing a data format

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

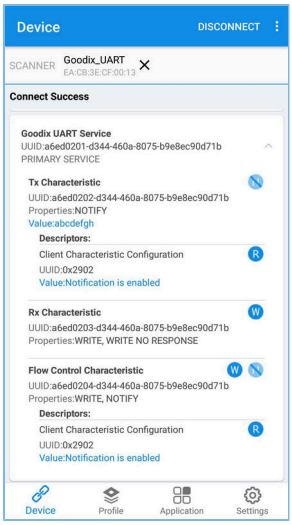

Figure 5-5 Presenting the string “abcdefgh”

6 Appendix: Throughput Test Result

Bluetooth LE throughput tests on Goodix SPP are performed based on the GR5515 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.

Table 6-1 Throughput in different modes

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