GR551x Bluetooth Low Energy Examples Application Manual

Version: 1.9

Release Date: 2022-02-20

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

Goodix provides a rich set of Bluetooth Low Energy (Bluetooth LE) application examples in a GR551x Software Development Kit (SDK). This document lists some examples with elaboration on code and verification methods in helping users to rapidly develop Bluetooth LE applications.

Audience

This document is intended for:

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

Release Notes

This document is the seventh release of *GR551x Bluetooth Low Energy Examples Application Manual*, corresponding to GR551x System-on-Chip (SoC) series.

Revision History

Version	Date	Description
1.0	2019-12-08	Initial release
1.3	2020-03-16	Added “DFU”.
1.5	2020-05-30	Updated description on Current Time Profile in “Current Time Server” and “Current Time Client”.
1.6	2020-06-30	Updated the document version based on SDK changes.
1.7	2021-05-20	Updated SoC model descriptions.
1.8	2021-11-01	Modified SoC model descriptions.
1.9	2022-02-20	• Modified the file name of the example firmware based on SDK changes. • Updated the “Preparation” section.

Contents

1 Introduction

To help users rapidly understand the APIs and implement Bluetooth Low Energy (Bluetooth LE) applications, the GR551x Software Development Kit (SDK) provides abundant Bluetooth LE application examples:

- Server, Client, and Broadcaster examples defined in *GATT Specifications*
- Goodix-defined GATT profile examples
- Other examples such as FreeRTOS and Apple Notification Center Service (ANCS)

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

Table 1-1 Reference documents

Name	Description
GR551x Developer Guide	Introduces GR551x SDK and how to develop and debug applications based on the SDK.
GR551x Bluetooth Low Energy Stack User Guide	Introduces the Bluetooth Low Energy Protocol Stack supported by GR551x System-on-Chips (SoCs).
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/.
Bluetooth Core Spec	Offers official Bluetooth standards and core specification from Bluetooth SIG.

2 Initial Operation

This chapter taking ble_app_template as an example, introduces how to use a BLE application example in the GR551x SDK.

Note:

SDK_Folder is the root directory of the GR551x SDK.

2.1 Preparation

Perform the following tasks before running a Bluetooth LE example.

- **Hardware preparation**

Table 2-1 Hardware preparation

Name	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 (SK Board; two boards are required for verification on some examples.)
Connection cable	Micro USB 2.0 serial cable
Android phone	Phones running on Android 5.0 (KitKat) and later versions
iOS device	An iOS device supporting Bluetooth LE v4.0 or later versions, such as iPhone 4s/iPad 3 or later versions

- **Software preparation**

Table 2-2 Software preparation

Name	Description
Windows	Windows 7/Windows 10
J-Link driver	A J-Link driver. Available at www.segger.com/downloads/jlink/.
GProgrammer (Windows)	A programming tool. Available in SDK_`Folder\tools\GProgrammer`.
GRUart (Windows)	A serial port debugging tool. Available in SDK_`Folder\tools\GRUart`.
GRToolbox (Android)	A Bluetooth LE debugging tool for GR551x. Available in SDK_`Folder\tools\GRToolbox`.

2.2 Firmware Programming

The source code of the ble_app_template example is in SDK_`Folder\projects\ble\ble_peripheral\ble_app_template`.

You can programme *ble_app_template.bin* to the SK Board through GProgrammer. For details, see *GProgrammer User Manual*.

Note:

- The *ble_app_template.bin* is in `SDK_Folder\projects\ble\ble_peripheral\ble_app_template\build`.

3 Bluetooth LE Examples Overview

You can find the Bluetooth LE application examples in SDK_Folder\projects\ble\. The examples have the following in common: main() function, peripheral initialization function, application initialization, GATT service event handlers, BLE SDK callback, and Sensor Simulator.

- main() function

  The main() function calls initialization functions of Bluetooth LE peripherals and BLE Stack, after which while(1) {} main loop runs. In this main loop, you can add event schedule code at the application layer (including code relative to handling user input and GUI flush). Sample example: UART Server. For details, see "Section 4.19 UART Server". The pwr_mgmt_schedule() function shall be called at the end of the loop to implement automatic power consumption management.

  A typical *main.c* file is described as below:

```
STACK_HEAP_INIT(heaps_table);

static app_callback_t app_ble_callback =
{
    .app_ble_init_cmp_callback  = ble_init_cmp_callback,
    .app_gap_callbacks   = &app_gap_callbacks,
    .app_gatt_common_callback   = &app_gatt_common_callback,
    .app_gattc_callback  = &app_gattc_callback,
    .app_sec_callback    = &app_sec_callback,
};

int main (void) {
    /*< Initialize user peripherals. */
    app_periph_init();
    /*< Initialize BLE Stack. */
    ble_stack_init(&app_ble_callback, &heaps_table);
    // Main Loop
    while(1)  {
         app_log_flush();

         pwr_mgmt_schedule();     }
}
```

  For details, see *GR551x Developer Guide*.

- Peripheral initialization function: app_periph_init()

  Set the power management mode after configuring the APP LOG module. You can initialize other hardware modules on demand.

  GR551x SDK simplifies setting Bluetooth device address (BD ADDR) by providing users with SYS_SET_BD_ADDR() through which you can set a temporary BD ADDR during product development. The code snippet is as follows:

```
/**@brief Bluetooth device address. */
static const uint8_t s_bd_addr[SYS_BD_ADDR_LEN] = {0x08, 0x00, 0xcf, 0x3e, 0xcb, 0xea};

void app_periph_init(void)
{
     SYS_SET_BD_ADDR(s_bd_addr);
     bsp_uart_init();
     app_log_assert_init();
```

```
    pwr_mgmt_mode_set(PMR_MGMT_SLEEP_MODE);
}
```

Note:

Remember to delete the SYS_SET_BD_ADDR() function before releasing official product code. For official releases, the BD ADDR is stored in eFuse. Therefore, the SYS_SET_BD_ADDR() function is no longer required for setting BD ADDR in flash memories. For details about eFuse, see *GR551x Datasheet*. GProgrammer allows writing data to eFuse.

- Application initialization

  Callback function of app_ble_init_cmp_cb_t type is called after initialization of the BLE Stack. You can implement application initialization in the callback function. Example operations include adding Bluetooth LE services, initializing GAP parameters, setting advertising parameters, and starting advertising.

  In the Bluetooth LE application examples, the callback function is implemented in the *user_app.c* file and registered in a GR551x SDK through the ble_stack_init() function. The code snippet is as follows:

```
void ble_init_cmp_callback(void)
{
    sdk_err_t     error_code;
    gap_bdaddr_t  bd_addr;
    sdk_version_t version;

    sys_sdk_verison_get(&version);
    APP_LOG_INFO("Goodix GR551x SDK V%d.%d (commit %d)",version.major,version.minor,
                 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("HID Mouse example started." );

    hw_simulator_init();
    error_code = app_timer_create(&s_hw_simulator_timer_id,ATIMER_REPEAT,
                                  hw_simulator_timer_handler);

    APP_ERROR_CHECK(error_code);
    error_code = app_timer_start(s_hw_simulator_timer_id, HW_SIM_UPDATE_INTERVAL, NULL);

    APP_ERROR_CHECK(error_code);
    services_init();
    gap_params_init();
    adv_params_init(true);
}
```

- GATT service event handlers

  Applications offer each of the GATT services an event handler through which the service can inform the applications about the events inside the service. For certain applications which are not interested in some GATT

service events, set the pointer of the corresponding event handler function to NULL to ignore the event. In the Bluetooth LE application examples, an event handler is named in the format: xxx_service_event_process. The handlers are defined in the *user_app.c* file.

- BLE SDK callback

  Use the ble_stack_init() function to register BLE SDK callbacks to a GR551x SDK, after which you can process Bluetooth LE events from the SDK in these callback functions. Example events include stopping advertising, disconnection, and updating connection parameters. These callbacks can be implemented in the source file under `Src\user_callback`. For details, see *GR551x Developer Guide*.

- Sensor Simulator

  Some GATT profiles are used to transmit data collected by sensors. The SK Board does not integrate a sensor, so the Bluetooth LE application examples adopt a sensor simulator. You can find a Sensor Simulator module in SDK_`Folder\components\libraries\sensorsim` in a GR551x SDK.

4 Bluetooth Peripheral Examples

This chapter introduces some common Bluetooth LE peripheral application examples.

4.1 Alert Notification Client

The Alert Notification Client example implements the Client role in the Alert Notification Profile (ANP). You can find the source code in SDK_Folder\projects\ble\ble_peripheral\ble_app_ans_c\. The Client connects to an Alert Notification Server, discovers Alert Notification Service, and obtains the Alert Notifications from the Server. Alert Notifications include incoming calls, missed calls, and SMS/MMS.

4.1.1 Code Interpretation

After the Alert Notification Client successfully connects to an Alert Notification Server and discovers Alert Notification Service from the Server, ans_c_new_alert_notify_set() is called followed by ans_c_unread_alert_notify_set() to enable All New Alert Category Characteristic and All Unread Alert Category Characteristic Notification on the Server.

Call ans_c_sup_new_alert_cat_read(), ans_c_sup_unread_alert_cat_read(), and ans_c_ctrl_point_set() to obtain notifications from the Server. Print the notifications on GRUart.

4.1.2 Test and Verification

You can test the Alert Notification Client through Alert Notification Server and GRUart by performing the steps below:

1. Download *ble_app_ans_c.bin* to SK Board 1 (as the Client) through GProgrammer.
2. Download *ble_app_ans.bin* to SK Board 2 (as the Server) through GProgrammer.
3. Connect the serial port of the Board 1 to the PC, start GRUart, and configure the serial port on the GRUart.
4. After the Client connects to the Server, press **OK** on the Board 1 to obtain New Alert information and Unread Alert information from the Server.
5. The Alert Notifications are displayed on the GRUart.

4.2 Beacon

The Beacon example conforms to iBeacon, an advertising protocol developed by Apple®. You can find the source code in SDK_Folder\projects\ble\ble_peripheral\ble_app_beacon\.

4.2.1 Code Interpretation

The iBeacon is based on standard Bluetooth LE protocols, so the Beacon example conforming to iBeacon advertises data with a type that is specified by manufacturers. The advertising data is defined in the following arrays:

```
static uint8_t s_adv_data_set[] =
{
    // Manufacturer specific adv data type
    0x1A,
    BLE_GAP_AD_TYPE_MANU_SPECIFIC_DATA,
    // Company Identifier
    0x4C, 0x00,
```

```
    // Beacon Data Type
    0x02,
    // Data Length
    0x15,
    // UUID - Variable based on different use cases/applications
    0x01, 0x12, 0x23, 0x34,
    0x45, 0x56, 0x67, 0x78,
    0x89, 0x9a, 0xab, 0xbc,
    0xcd, 0xde, 0xef, 0xf0,
    // Major value for identifying Beacons
    0x00,
    0x01,
    // Minor value for identifying Beacons
    0x00,
    0x01,
    // The Beacon's measured RSSI at meter distance in dBm
    0xc3
}
```

After initializing GAP parameters and advertising data, the Beacon example starts a five-second timer to update the Major and Minor values in the advertising data.

4.2.2 Test and Verification

You can test the Beacon example through GRToolbox by performing the steps below:

1. Download *ble_app_beacon.bin* to an SK Board through GProgrammer.
2. Scan nearby Bluetooth LE devices on GRToolbox, and find the iBeacon device whose BD ADDR is the same as that displayed on GRUart.
3. You can view the Major and Minor values of the advertising data on GRToolbox.

Note:

The Major and Minor values change dynamically as time passes by to allow users to easily view the operating state of the application. Such changes are not regulated in the iBeacon protocol.

4.3 Blood Pressure Sensor

The Blood Pressure Sensor example implements the Sensor role in the Blood Pressure Profile (BLP). You can find the source code in SDK_Folder\projects\ble\ble_peripheral\ble_app_bps\.

The example includes the two GATT services in the BLP:

- Blood Pressure Service
- Device Information Service

The example also includes a GATT service that is not regulated in the BLP:

- Battery Service

The Blood Pressure Service supports the following three characteristics:

- Blood Pressure Measurement

- Blood Pressure Measurement Client Characteristic Configuration descriptor
- Blood Pressure Feature

Use the Sensor Simulator module in a GR551x SDK to simulate the changing characteristic values in the Blood Pressure Service and Battery Service.

4.3.1 Code Interpretation

The Blood Pressure Sensor supports initiating up to five concurrent extended advertisements. It can connect to and transmit data to up to 10 collectors concurrently.

The start_next_adv() function decides whether to start a new advertisement.

- When a new advertisement is initiated, the start_next_adv() function is called by app_gap_adv_start_cb().
- When a new advertisement is stopped because a connection is established between the Sensor and a Collector, the start_next_adv() function is called by app_gap_adv_stop_cb().
- When a Collector is disconnected from a Sensor, the start_next_adv() function is called by app_disconnect_handler().

In scenarios where 10 connections have been established, the Blood Pressure Sensor stops initiating new advertisements.

The Blood Pressure Sensor supports Just Works pairing. After the Sensor pairs with and bonds to a Collector, the Collector can enable Blood Pressure Measurement Indication on the Sensor. The timeout handler function, bps_sim_measurement(), of the Blood Pressure Measurement Timer reassembles simulated data of the Blood Pressure Measurement into packets and transfers the data.

4.3.2 Test and Verification

You can test the Blood Pressure Sensor through GRToolbox by performing the steps below:

1. Download *ble_app_bps.bin* to an SK Board through GProgrammer.
2. In **Profile** > **BPM** on GRToolbox, enable device scanning and connect to a device advertising as "Goodix_BLP".
3. After successful connection, you can view the changing blood pressure data in the BPM module: Systolic, Diastolic, Mean AP, and Pulse Rate.

4.4 Current Time Server

The Current Time Server example implements the Server role of the Current Time Profile. You can find the source code in `SDK_Folder\projects\ble\ble_peripheral\ble_app_cts\`.

The Current Time Server includes the Current Time Service that supports the following three characteristics:

- Current Time
- Local Time Information
- Reference Time Information

4.4.1 Code Interpretation

A timer starts counting immediately after the Current Time Server is enabled. The timeout handler function, current_time_update(), of the timer continuously modifies the current time. If the Current Time Client enables the Current Time Characteristic Notification on the Server, the Server calls cts_cur_time_send() to notify the Client of the current time changes. The Client can also set the Current Time Information characteristic.

4.4.2 Test and Verification

Testing the Current Time Server requires associative working of a Current Time Client. For details about the testing methods, see "Section 5.2 Current Time Client".

4.5 Cycling Speed and Cadence Sensor

The Cycling Speed and Cadence Sensor example implements the Sensor role in the Cycling Speed and Cadence Profile (CSCP). You can find the source code in `SDK_Folder\projects\ble\ble_peripheral\ble_app_cscs\`.

The Cycling Speed and Cadence Sensor includes three GATT services:

- Cycling Speed and Cadence Service (mandatory for CSCP)
- Device Information Service (optional for CSCP)
- Battery Service (not regulated in CSCP)

The CSCP supports the following six characteristics:

- CSC Measurement
- CSC Measurement Client Characteristic Configuration descriptor
- CSC Feature
- Sensor Location
- SC Control Point
- SC Control Point Client Characteristic Configuration Descriptor

Use the Sensor Simulator module in a GR551x SDK to simulate the changing characteristic values in the Cycling Speed and Cadence Service and Battery Service.

4.5.1 Code Interpretation

After a Cycling Speed and Cadence Collector enables CSC Measurement Characteristic Notification on a Cycling Speed and Cadence Sensor, the CSC Measurement timer starts to count. The timeout handler function, csc_meas_timeout_handler(), of the timer is executed periodically to simulate the changing parts of the CSC Measurement characteristic value:

- Cumulative Wheel Revolutions
- Last Wheel Event Time
- Cumulative Crank Revolutions

- Last Crank Event Time

Afterwards, the Sensor calls the Cycling Speed and Cadence Service API, csc_measurement_send(), to send the CSC Measurement characteristic value to the Cycling Speed and Cadence Collector.

4.5.2 Test and Verification

You can test the Cycling Speed and Cadence Sensor through GRToolbox by performing the steps below:

1. Download *ble_app_cscs.bin* to an SK Board through GProgrammer.
2. In **Profile** > **CSC** on GRToolbox, enable device scanning and connect to a device advertising as "Goodix_CSCS".
3. After successful connection, you can view the changing cycling speed and cadence data in the CSC module.

4.6 Glucose Sensor

The Glucose Sensor example implements the Sensor role in the Glucose Profile (GLP). You can find the source code in `SDK_Folder\projects\ble\ble_peripheral\ble_app_gls\`.

The example includes the two GATT services in the GLP:

- Glucose Service
- Device Information Service

The example also includes a GATT service that is not regulated in the GLP:

- Battery Service

The Glucose Service supports the following five characteristics:

- Glucose Measurement
- Glucose Measurement Client Characteristic Configuration descriptor
- Glucose Feature
- Record Access Control Point
- Record Access Control Point Client Characteristic Configuration Descriptor

Use the Sensor Simulator module in a GR551x SDK to simulate the changing characteristic value in the Battery Service. Press **OK** on the SK Board to control the Sensor to create a Glucose Measurement characteristic value in the Glucose Service.

4.6.1 Code Interpretation

The Glucose Collector enables Glucose Measurement Characteristic Notification on the Sensor. If the Collector writes related Glucose Database instructions to the Record Access Control Point characteristic, the Sensor calls gls_meas_val_send() to send the Glucose Measurement value to the Collector. The instructions include:

- Sequence Number
- Base Time & Time offset

- Glucose Concentration
- Sample Type
- Sample Location

Press **OK** on the SK Board, which enables the key_click_event_handler() function to call glucose_measurement_excute() to create an entry of Glucose sample data which is stored in the Glucose Database.

4.6.2 Test and Verification

You can test the Glucose Sensor through GRToolbox by performing the steps below:

1. Download *ble_app_gls.bin* to an SK Board through GProgrammer.
2. Enable scanning on GRToolbox, and tap and connect to the device advertising as “Goodix_GLS”.
3. Press **OK** on the SK Board to create a Glucose Measurement value.
4. Enable Glucose Measurement Notification on the Sensor, and write database obtaining instructions to Record Access Control Point. After these, the Collector receives the Glucose Measurement characteristic value.

4.7 Heart Rate Sensor

The Heart Rate Sensor example implements the Sensor role in the Heart Rate Profile (HRP). You can find the source code in SDK_Folder\projects\ble\ble_peripheral\ble_app_hrs\.

The example includes the two GATT services in the HRP:

- Heart Rate Service
- Device Information Service

The example also includes a GATT service that is not regulated in the HRP:

- Battery Service

The Heart Rate Service supports the following four characteristics:

- Heart Rate Measurement
- Heart Rate Measurement Client Characteristic Configuration descriptor
- Body Sensor Location
- Heart Rate Control Point

Use the Sensor Simulator module in a GR551x SDK to simulate the changing characteristic values in the Heart Rate Service and Battery Service.

4.7.1 Code Interpretation

After the Heart Rate Collector enables Heart Rate Measurement Characteristic Notification on the Sensor, the Heart Rate Measurement Timer and RR Interval Measurement Timer begin to count. The timeout handler function,

heart_rate_meas_timeout_handler(), of the Heart Rate Measurement Timer simulates the changing parts of the Heart Rate Measurement characteristic value.

- Heart Rate
- Energy Expended
- Sensor Contact Detected

The heart_rate_meas_timeout_handler() function calls hrs_heart_rate_measurement_send() to send notifications. The Heart Rate Collector receives heart rates changing in triangular waves. The Sensor Contact status dynamically switches between Detected and Undetected.

The timeout handler function, rr_interval_timeout_handler(), of the RR Interval Measurement Timer simulates the changing RR Interval. The rr_interval_timeout_handler() function sends the measured RR Interval to Heart Rate Service for storage. The function does not push notifications.

4.7.2 Test and Verification

You can test the Heart Rate Sensor through GRToolbox by performing the steps below:

1. Download *ble_app_hrs.bin* to an SK Board through GProgrammer.
2. In **Profile** > **HRS** on GRToolbox, enable device scanning and connect to a device advertising as "Goodix_HRM".
3. After successful connection, you can view the heart rate value and curve, RR Interval, as well as Location Finger in the HRS module.

4.8 Heart Rate Sensor Multi Link

Similar to Heart Rate Sensor, the Heart Rate Sensor Multi Link example implements the Sensor role in the Heart Rate Profile (HRP). The Sensor connects to and provides heart rates for multiple Heart Rate Collectors. You can find the source code in SDK_Folder\projects\ble\ble_peripheral\ble_app_hrs_multi_link\.

4.8.1 Code Interpretation

Enable the Heart Rate Sensor Multi Link example to start advertising. After the Sensor connects to a Heart Rate Collector, multi_advertising_start() is called to restart advertising on the Sensor. When the established connection quantity reaches the maximum configurable threshold, no advertising is started.

4.8.2 Test and Verification

You can test the Heart Rate Sensor Multi Link through GRToolbox by performing the steps below:

1. Download *ble_app_hrs_multi_link.bin* to an SK Board through GProgrammer.
2. In **Profile** > **HRS** on GRToolbox, enable device scanning and connect to a device advertising as "GR_HRM_MLINK".
3. After successful connection, you can view the changing heart rate curve and Contact Detected status in the HRS module.

4. Use another phone to repeat Steps 2 and 3. A multi-link connection layout is established. After successful connection, you can view the changing heart rate curve and Contact Detected status in the HRS module.

4.9 Phone Alert Status Client

The Phone Alert Status Client example implements the Client role in the Phone Alert Status Profile (PASP). You can find the source code in SDK_Folder\projects\ble\ble_peripheral\ble_app_pass_c\. After the Client connects to a Phone Alert Status Server and discovers the Phone Alert Status Service, the Client obtains the alert status and ring settings from the Server.

4.9.1 Code Interpretation

After the Phone Alert Status Client successfully connects to a Phone Alert Status Server and discovers Phone Alert Notification Service from the Server, pass_c_alert_status_notify_set() is called followed by pass_c_ringer_set_notify_set() to enable Alert Status and Ring Set Notification on the Server. Call pass_c_ctrl_point_set() to set the ring on the Server.

4.9.2 Test and Verification

You can test the Phone Alert Status Client in association with Phone Alert Status Server through GRToolbox by performing the steps below:

1. Download *ble_app_pass_c.bin* to SK Board 1 (as the Client) through GProgrammer.
2. Download *ble_app_pass.bin* to SK Board 2 (as the Server) through GProgrammer.
3. Connect the serial port of the Board 1 to the PC, start GRUart, and configure the serial port on the GRUart.
4. After the Client connects to the Server, press **OK** on the Board 1 to set the Server in Silent mode; double-press **OK** to set the Server in Mute Once mode; long-press **OK** to cancel the Silent mode.
5. GRUart displays the Alert Status information received from the Server on the Client. You can view the details: **Ringer State**, **Vibrate State**, and **Display State**.

4.10 Proximity Reporter

The Proximity Reporter example implements the Reporter role in the Proximity Profile (PXP). You can find the source code in SDK_Folder\projects\ble\ble_peripheral\ble_app_pxp_reporter\.

The example includes the three GATT services in the PXP:

- Link Loss Service
- Immediate Alert Service
- TX Power Service

4.10.1 Code Interpretation

According to *Proximity Profile (PXP) Specification*, the Proximity Reporter initiates fast connection advertising within 30 seconds upon being powered on and switches to reduced power advertising after 30 seconds upon being powered

on. The app_gap_op_cmp_evt_cb() function in the *user_gap_callback.c* file calls app_adv_stopped_handler() to initiate reduced power advertising.

The Monitor initiates a connection request to the Reporter and receives a pairing request from the Reporter. The app_sec_rcv_enc_req_cb() function in the *user_sm_callback.c* file handles the pairing and encryption requests from the BLE Stack. In the TK_REQ function branch, the statement **tk = 123456** sets the pin code as **123456**.

4.10.2 Test and Verification

1. Download *ble_app_pxp_reporter.bin* to an SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart.
3. Enable scanning on GRToolbox, and tap and connect to the device advertising as “Goodix_PXP”.
4. Enter **123456** in **Pin Code** on the phone to complete pairing and bonding.
5. Take the phone far away from the board until the phone is disconnected from the board.
6. You can view the Link Loss Alert Message in the **Receive Data** pane on GRUart.

4.11 Running Speed and Cadence Sensor

The Running Speed and Cadence Sensor example implements the Sensor role in the Running Speed and Cadence Profile (RSCP). You can find the source code in SDK_`Folder\projects\ble\ble_peripheral\ble_app_rscs\`.

The example includes the following GATT services:

- Running Speed and Cadence Service (mandatory for RSCP)
- Device Information Service (optional for RSCP)
- Battery Service (not regulated in RSCP)

The Running Speed and Cadence Service supports the following six characteristics:

- RSC Measurement
- RSC Measurement Client Characteristic Configuration descriptor
- RSC Feature
- Sensor Location
- SC Control Point
- SC Control Point Client Characteristic Configuration Descriptor

Use the Sensor Simulator module in a GR551x SDK to simulate the changing characteristic values in the Running Speed and Cadence Service and Battery Service.

4.11.1 Code Interpretation

After a Running Speed and Cadence Collector enables RSC Measurement Characteristic Notification on a Running Speed and Cadence Sensor, the RSC Measurement timer starts to count. The timeout handler function,

rsc_meas_update(), of the timer is executed periodically to simulate the changing parts of the RSC Measurement characteristic value:

- Instantaneous Speed
- Instantaneous Cadence
- Instantaneous Stride Length
- Total Distance
- Walking or Running Status

Afterwards, call the Running Speed and Cadence Service API, rsc_measurement_send(), to send the RSC Measurement characteristic value to the Running Speed and Cadence Collector.

4.11.2 Test and Verification

You can test the Running Speed and Cadence Sensor through GRToolbox by performing the steps below:

1. Download *ble_app_rscs.bin* to an SK Board through GProgrammer.
2. In **Profile** > **RSC** on GRToolbox, enable device scanning and connect to a device advertising as "Goodix_RSCS".
3. After successful connection, you can view the changing running speed and cadence data in the RSC module.

4.12 Thermometer Sensor

The Thermometer Sensor example implements the Thermometer role in the Health Thermometer Profile (HTP). You can find the source code in SDK_Folder\projects\ble\ble_peripheral\ble_app_hts\.

The example includes the two GATT services in the HTP:

- Health Thermometer Service
- Device Information Service

The example also includes a GATT service that is not regulated in the HTP:

- Battery Service

The Health Thermometer Service supports the following eight characteristics:

- Temperature Measurement
- Temperature Measurement Client Characteristic Configuration descriptor
- Temperature Type
- Intermediate Temperature
- Intermediate Temperature Client Characteristic Configuration descriptor
- Measurement Interval
- Measurement Interval Client Characteristic Configuration descriptor
- Measurement Interval Valid Range descriptor

Use the Sensor Simulator module in a GR551x SDK to simulate the changing characteristic values in the Health Thermometer Service and Battery Service.

4.12.1 Code Interpretation

Use gap_params_init() to enable pairing, and call ble_gap_privacy_params_set() to enable privacy mode, which means a random address is updated every 900 seconds. The Thermometer Sensor continuously initiates advertising until it receives a connection request from a Collector.

When the Thermometer Sensor connects to a Collector, the Battery Level Timer starts counting to simulate the battery level changes. When the Collector enables Temperature Measurement Indication or Intermediate Temperature Measurement Notification on the Thermometer, the Temperature Level Timer starts counting. Afterwards, the hts_meas_timeout_handler() is executed periodically to simulate the temperature level changes.

When the Thermometer Sensor disconnects from the Collector, app_disconnect_handler() is called to stop the Battery Level Timer and Temperature Level Timer.

4.12.2 Test and Verification

You can test the Thermometer Sensor through GRToolbox by performing the steps below:

1. Download *ble_app_hts.bin* to an SK Board through GProgrammer.
2. In **Profile** > **HTS** on GRToolbox, enable device scanning and connect to a device advertising as "Goodix_HTS".
3. After successful connection, you can view the changing temperatures in the HTS module with a time increment interval of 2 seconds.

4.13 Throughput Server

The Throughput Server example implements the Server role in the Goodix Throughput Profile. You can find the source code in SDK_Folder\projects\ble\ble_peripheral\ble_app_throughput\. The example includes Goodix-defined Throughput Service. It demonstrates the throughput during Bluetooth LE data transmission. The Throughput Server does not output throughput and related parameters by using local physical serial ports. Instead, you can view the real-time throughput and related parameters in the Throughput module on GRToolbox.

4.13.1 Code Interpretation

The gap_params_init() function in the *user_app.c* file sets the Preferred Data Length, MTU, and PHY that impact the throughput. You can modify these parameters on GRToolbox after the Throughput Server connects to a Client. The ths_event_process() function in the *throughput.c* file responds to the parameter changes in the THS_EVT_SETTINGS_CHANGED branch.

In addition, the ths_event_process() function counts the total amount of received data in the THS_EVT_DATA_RECEIVED branch. THS_EVT_TOGGLE_SET and THS_EVT_DATA_SENT are used to transmit data in the ths_event_process() function. If the THS_EVT_TOGGLE_SET is set to THS_TOGGLE_STATE_ON, ths_send_data() is called to transmit the first data packet. The next data packet is delivered when the Server receives THS_EVT_DATA_SENT. When the received THS_EVT_TOGGLE_SET is set to THS_TOGGLE_STATE_OFF, data packet delivery is stopped.

4.13.2 Test and Verification

You can test the Throughput Server through GRToolbox by performing the steps below:

1. Download *ble_app_throughput.bin* to an SK Board through GProgrammer.
2. Enable scanning on GRToolbox, and tap and connect to the device advertising as “Goodix_THS”.
3. In **Application** > **THS** on GRToolbox, configure Data Length, MTU, PHY, TX Power, and CI.
4. In **Application** > **THS** on GRToolbox, tap **Toggle On** to enable the board to transmit data.
5. In **Application** > **THS** on GRToolbox, you can view the real-time throughput curve of data transmitted from the board to the phone. Tap **Toggle Off** to disable data transmission.
6. In **Application** > **THS** on GRToolbox, set the TX mode as Master Write, which allows the phone to transmit data to the board. You can view the real-time throughput curve on GRToolbox.

4.14 ANCS Client

The ANCS Client example implements the Client role in the Apple Notification Center Service and demonstrates how a GR551x SoC communicates and interacts with an iOS device through ANCS. You can find the source code in SDK_Folder\projects\ble\ble_peripheral\ble_app_ancs_c. For details, see *GR551x ANCS Profile Example Application*.

4.15 FreeRTOS Template

The FreeRTOS Template example supports multi-task scheduling by adopting FreeRTOS. You can find the source code in SDK_Folder\projects\ble\ble_peripheral\ble_app_template_freertos. For details, see *GR551x FreeRTOS Example Application*.

4.16 HID Mouse

The HID Mouse example implements the HID mouse in the HID Over GATT Profile. You can find the source code in SDK_Folder\projects\ble\ble_peripheral\ble_app_hids_mouse. For details, see *GR551x HID Mouse Example Application*.

4.17 Power Consumption

The Power Consumption example implements the Goodix-defined Power Consumption Profile and helps test the power consumption of GR5515 SK Boards. You can find the source code in SDK_Folder\projects\ble\ble_peripheral\ble_app_pcs. For details, see *GR551x Power Consumption Profile Example Application*.

4.18 Template

The Template example implements the Server role of the Goodix-defined Sample Service. You can find the source code in SDK_Folder\projects\ble\ble_peripheral\ble_app_template. For details, see *GR551x Sample Service Application and Customization*.

4.19 UART Server

The UART Server example implements the Goodix Serial Port Profile (SPP), allowing users to rapidly develop Bluetooth LE applications based on data passthrough. You can find the source code in SDK_Folder\projects\ble\ble_peripheral\ble_app_uart. For details, see *GR551x Serial Port Profile Example Application*.

4.20 AMS Client

The AMS Client example implements the Client role in the Apple Media Service and demonstrates how a GR551x SoC communicates and interacts with an iOS device through AMS. You can find the source code in SDK_Folder\projects\ble\ble_peripheral\ble_app_ams_c. For details, see *GR551x AMS Profile Example Application*.

5 Bluetooth LE Central Examples

5.1 Alert Notification Server

The Alert Notification Server example implements the Server role in the Alert Notification Profile (ANP). You can find the source code in SDK_Folder\projects\ble\ble_central\ble_app_ans\.

The Alert Notification Service supports the following five characteristics:

- Supported New Alert Category
- New Alert
- Supported Unread Alert Category
- Unread Alert Status
- Alert Notification Control Point

5.1.1 Code Interpretation

Start the Alert Notification Server, and press **OK** on the GR5515 SK Board to simulate Unread Alerts such as New Email and Missed Call. Call ans_unread_alert_send() to send Unread Alert and ans_new_alert_send() to send New Alert to the Alert Notification Client. The Alert Notification includes:

- Alert ID
- Alert Number

Note:

Alert ID and Alert Number are not displayed on GRUart.

5.1.2 Test and Verification

For test method details, see "Section 4.1 Alert Notification Client".

5.2 Current Time Client

The Current Time Client example implements the Client role of the Current Time Profile. You can find the source code in SDK_Folder\projects\ble\ble_central\ble_app_cts_c\. After the Client connects to a Current Time Server with the Current Time Service discovered, the Client can set and obtain the current time information of the Server.

5.2.1 Code Interpretation

After the Current Time Client successfully connects to a Current Time Server and discovers Current Time Service from the Server, cts_c_cur_time_notify_set() is called to enable Current Time characteristic Notification on the Server. Use **OK** on a GR5515 SK Board to call cts_c_cur_time_read(), cts_c_loc_time_info_read(), and cts_c_ref_time_info_read() to obtain the current time, local time information, and reference time information of the Server. You can view the information printed on GRUart.

5.2.2 Test and Verification

You can test the Current Time Client through Current Time Server and GRUart by performing the steps below:

1. Download *ble_app_cts_c.bin* to SK Board 1 (as the Client) through GProgrammer.
2. Download *ble_app_cts.bin* to SK Board 2 (as the Server) through GProgrammer.
3. Connect the serial port of the Board 1 to the PC, start GRUart, and configure the serial port on the GRUart.
4. After the Client connects to the Server, press **OK** on the Board 1 to obtain current time, local time, and reference time information from the Server.
5. The current time information is displayed on the GRUart.

5.3 Phone Alert Status Server

The Phone Alert Status Server example implements the Server role in the Phone Alert Status Profile (PASP). You can find the source code in SDK_Folder\projects\ble\ble_central\ble_app_pass\.

The Phone Alert Status Service implemented in this example supports the following three characteristics:

- Alert Status
- Ringer Setting
- Ringer Control Point

5.3.1 Code Interpretation

When the Phone Alert Status Client enables Alert Status and Ringer Setting Characteristic Notification on the Server, any change on Alert Status and Ringer Setting Characteristic Notification can trigger pass_ringer_setting_set() and pass_alert_status_set() to send the current Ringer Setting and Alert Status to the Client.

5.3.2 Test and Verification

For test method details, see “Section 4.9 Phone Alert Status Client”.

5.4 Heart Rate Collector

The Heart Rate Collector example implements the Collector role in the Heart Rate Profile (HRP). After the Collector connects to a Heart Rate Sensor, the Collector receives detected heart rate information from the Sensor. You can find the source code in SDK_Folder\projects\ble\ble_central\ble_app_hrs_c\. The Collector example supports interaction with Heart Rate Service, Device Information Service, and Battery Service.

5.4.1 Code Interpretation

Start the Collector, and enable device scanning. Use hrs_srvc_uuid_find() to check whether the scanned advertising data contains a Heart Rate Service UUID. If yes, the scanned device is regarded as a target device to which the Collector initiates a connection request.

After successful connection, use the **OK** button on an SK Board to enable or disable Heart Rate Measurement Characteristic Notification and Battery Level Characteristic Notification.

When both Heart Rate Measurement Characteristic Notification and Battery Level Characteristic Notification are enabled, the received heart rate data is reported to GRUart through HRS_C_EVT_HR_MEAS_VAL_RECEIVE event in hrs_c_evt_process(). The battery level data is reported to the Collector through BAS_C_EVT_BAT_LEVE_RECEIVE event in bas_c_evt_process(). You can view the information on GRUart.

5.4.2 Test and Verification

1. Download *ble_app_hrs_c.bin* to an SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart.
3. GRUart prints device boot, scanning, and connection logs.
4. Press **OK** on the SK Board to enable HRS notification and BAS notification.
5. You can view the heart rate and battery level of the Heart Rate Sensor on GRUart.

5.5 Running Speed and Cadence Collector

The Running Speed and Cadence Collector example implements the Collector role in the Running Speed and Cadence Profile (RSCP). After the Collector connects to a Running Speed and Cadence Sensor, the Collector receives detected running speed and cadence information from the Sensor. You can find the source code in `SDK_Folder\projects\ble\ble_central\ble_app_rscs_c\`. The Collector example supports interaction with Running Speed and Cadence Service, Device Information Service, and Battery Service.

5.5.1 Code Interpretation

Start the Collector, and enable device scanning. Use rscs_srvc_uuid_find() to check whether the scanned advertising data contains a Running Speed and Cadence Service UUID. If yes, the scanned device is regarded as a target device to which the Collector initiates a connection request.

After successful connection, use the **OK** button on an SK Board to enable or disable RSC Measurement Characteristic Notification and Battery Level Characteristic Notification.

When both RSC Measurement Characteristic Notification and Battery Level Characteristic Notification are enabled, the received running speed and cadence data is reported to GRUart through RSCS_C_EVT_RSC_MEAS_VAL_RECEIVE event in rscs_c_evt_process(). The battery level data is reported to the Collector through BAS_C_EVT_BAT_LEVE_RECEIVE event in bas_c_evt_process(). You can view the information on GRUart.

5.5.2 Test and Verification

1. Download *ble_app_rscs_c.bin* to an SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart.
3. GRUart prints device boot, scanning, and connection logs.
4. Press **OK** on the SK Board to enable RSCS notification and BAS notification.
5. You can view the running speed and cadence and battery level of the Running Speed and Cadence Sensor on GRUart.

5.6 UART Initiator

The UART Initiator example implements the Initiator role in the Goodix UART Profile. You can find the source code in `SDK_Folder\projects\ble\ble_central\ble_app_uart_c\`. When the Initiator connects to a UART Server (Acceptor), two-way transmission is allowed: The Initiator receives local UART data and transmits the data to the Acceptor through a Bluetooth LE connection; the Initiator receives Bluetooth LE data from the Acceptor and transmits the data to a local UART.

5.6.1 Code Interpretation

Use app_periph_init() to initialize the Initiator and ble_init_cmp_calback() to initialize the two-way ring buffer.

The UART data of the Initiator is received through hal_uart_rx_cplt_callback() and hal_uart_dma_rx_tfr_callback(). Call the uart_to_ble_push() function to write the UART data to s_uart_rx_ring_buffer (ring-shaped buffer from UART to BLE).

The Bluetooth LE data of the Acceptor is received in GUS_C_EVT_PEER_DATA_RECEIVE in gus_c_evt_process(). Call the ble_to_uart_push() function to write the UART data to s_ble_rx_ring_buffer (ring-shaped buffer from BLE to URAT).

Call uart_transport_schedule() API in the main loop in the *main.c* file to schedule the two-way transmission.

5.6.2 Test and Verification

1. Download *ble_app_gus_c.bin* to an SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart.
3. GRUart prints connection logs and information of CCCD enablement on the Acceptor.
4. Write data to the peer board (Acceptor) through GRUart. The Acceptor receives the data, and GRUart prints the data.
5. You can view the data of the Acceptor on GRUart.

6 Other Examples

6.1 BLE Basics

The BLE Basics include examples related to the BLE Stack. You can find the source code in SDK_Folder\projects\ble\ble_basic_example. For details, see *GR551x Bluetooth Low Energy Stack User Guide*.

6.2 BLE Multi Role

The BLE Multi Role example implements the Central and Peripheral roles. It supports connection to multiple devices at a time. You can find the source code in SDK_Folder\projects\ble\ble_multi_role\ble_app_hrs_rscs_relay. For details, see *GR551x HRS RSCS Relay Example Application*.

6.3 DFU

The DFU example demonstrates the methods of Device Firmware Update (DFU) for GR551x SoCs through GRToolbox (Android). You can find the source code in SDK_Folder\projects\ble\dfu. For details, see *GR551x OTA Example Application*.

6.4 DTM

The DTM example demonstrates how to use a Bluetooth tester to measure the radio frequency performance of GR551x SoCs in DTM mode. You can find the source code in SDK_Folder\projects\ble\dtm. For details, see *GR551x DTM Test Application Note*.