GR551x HRS RSCS Relay Example Application

Version: 1.8

Release Date: 2020-12-15

Shenzhen Goodix Technology Co., Ltd.

Trademarks and Permissions

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 a Heart Rate Sensor & Running Speed and Cadence Sensor Relay (HRS RSCS Relay) 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 sixth release of *GR551x HRS RSCS Relay 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	Adjusted the indentation of the code in “Chapter 4 Application Details”.
1.6	2020-06-30	Updated the document version based on SDK changes.
1.7	2020-11-09	Updated Figure 3-2 in “Section 3.4 Test and Verification”.
1.8	2020-12-15	Updated GRToolbox UI figure based on software update.

Contents

1 Introduction

The Heart Rate Sensor & Running Speed and Cadence Sensor Relay (HRS RSCS Relay) example demonstrates how to apply GR551x SoCs in scenarios with multi-roles (Peripheral and Central) and multi-connections, to enable functions of an HRS RSCS Relay device. The HRS RSCS Relay device can serve as both a collector and a sensor.

- Collector

  As a GATT Client, the HRS RSCS Relay device receives measurement data from heart rate sensor as well as running speed and cadence sensor.

- Sensor

  As a GATT Server, the HRS RSCS Relay device sends the received data to other collectors, such as GRToolbox (a Bluetooth LE debugging App for GR551x).

This document introduces how to use and verify an HRS RSCS Relay example in a GR551x SDK. Before you get started, it is recommended to refer to the following documents.

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	Introduces the software/hardware and quick start guide of GR551x SoCs.
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

The HRS RSCS Relay example implements the following profiles:

- Standard profiles: Heart Rate Profile as well as Running Speed and Cadence Profile, which are defined by Bluetooth SIG
- Custom profile: Goodix HRS RSCS Relay Control Point Profile, which is defined by Goodix

The application scenarios where GRToolbox is used as an HRS RSCS Relay collector are shown in Figure 2-1.

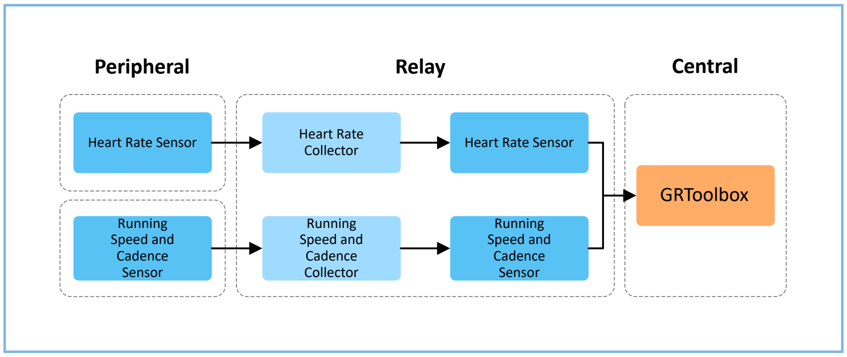

Figure 2-1 Application scenarios

HRS RSCS Relay device registers the following profiles when it is used as a collector:

- Heart Rate Client Profile: Receive measurement data from a heart rate sensor.
- Running Speed and Cadence Client Profile: Receive measurement data from a running speed and cadence sensor.

HRS RSCS Relay device registers the following profiles when it is used as a sensor:

- Heart Rate Server Profile: Relay the received data from a heart rate sensor to GRToolbox.
- Running Speed and Cadence Server Profile: Relay the received data from a running speed and cadence sensor to GRToolbox.
- Goodix HRS RSCS Relay Control Point Profile: Receive control commands from GRToolbox and returns execution outcomes.

Goodix HRS RSCS Relay Control Point Profile includes HRS RSCS Relay Control Point Service (HRRCPS), with a 128-bit vendor-specific UUID of A6ED0601-D344-460A-8075-B9E8EC90D71B.

HRRCPS has the following characteristics:

- HRR Control Point characteristic: Receive control commands from the HRS RSCS Relay collector.
- HRR Control Point Response characteristic: Return execution outcomes to the HRS RSCS Relay collector.

These characteristics are described in detail as follows:

Table 2-1 HRRCPS characteristics

Characteristic	UUID	Type	Support	Security	Property
HRR Control Point	A6ED0602-D344-460A-8075-B9E8EC90D71B	128 bits	Mandatory	None	Write
HRR Control Point Response	A6ED0603-D344-460A-8075-B9E8EC90D71B	128 bits	Mandatory	None	Indicate

3 Initial Operation

This chapter introduces how to quickly verify an HRS RSCS Relay example in a GR551x SDK.

Note:

SDK_Folder is the root directory of GR551x SDK.

3.1 Preparation

Perform the following tasks before running the example.

- **Hardware preparation**

Table 3-1 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 (GR5515 SK Board) (3 boards in total)
Cable	Micro USB 2.0 cable

- **Software preparation**

Table 3-2 Software preparation

Name	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 Bluetooth LE debugging tool for GR551x. Available in `SDK_Folder\tools\GRToolbox`.
GProgrammer (Windows)	A GR551x programming tool. Available in `SDK_Folder\tools\GProgrammer`.

3.2 Hardware Connection

To verify an HRS RSCS Relay example, use three development boards as the Relay device, the HRS device, and the RSCS device respectively. Connect three boards through Bluetooth LE.

Connect a GR5515 Starter Kit Board to a PC with a Micro USB 2.0 cable, as shown in Figure 3-1.

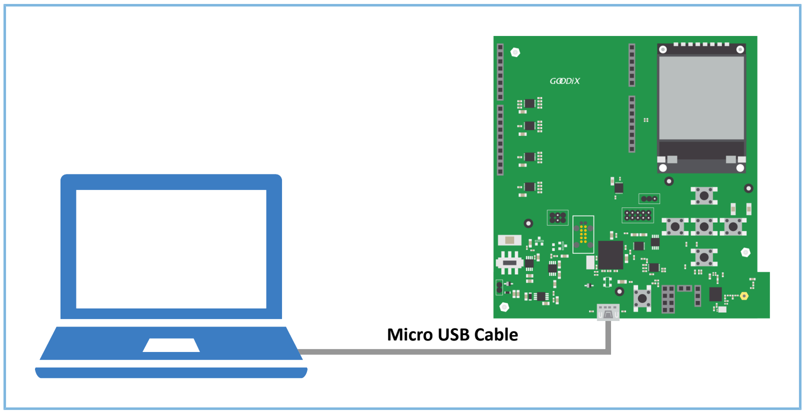

Figure 3-1 Hardware connection

3.3 Firmware Download

Download *ble_app_hrs_rscs_relay_fw.bin* to the Relay device, *ble_app_hrs_fw.bin* to the HRS device, and *ble_app_rscs_fw.bin* to the RSCS device.

For details on downloading firmware to the GR5515 SK Boards, see *GProgrammer User Manual*.

Note:

- The *ble_app_hrs_rscs_relay_fw.bin* is in `SDK_Folder\projects\ble\ble_multi_role\ble_app_hrs_rscs_relay\build`.
- The *ble_app_hrs_fw.bin* is in `SDK_Folder\projects\ble\ble_peripheral\ble_app_hrs\build`.
- The *ble_app_rscs_fw.bin* is in `SDK_Folder\projects\ble\ble_peripheral\ble_app_rscs\build`.

SDK_Folder is the root directory of GR551x SDK.

3.4 Test and Verification

When the HRS RSCS Relay device, the HRS device, and the RSCS device are ready, test and verify the HRS RSCS Relay example. Steps are described as follows:

1. Scan the HRS RSCS Relay device.

   Run GRToolbox, and select **Application** > **RELAY**.

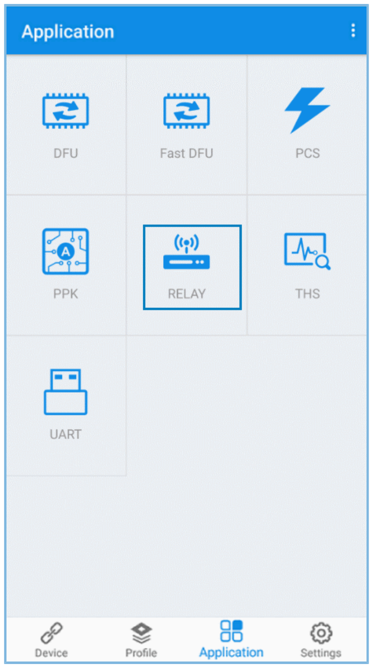

Figure 3-2 Choosing **RELAY**

Start scanning and discover a device with the advertising name **Goodix_HRS_RSCS_RELAY** (the advertising name can be modified in *user_app.c*), as shown in Figure 3-3.

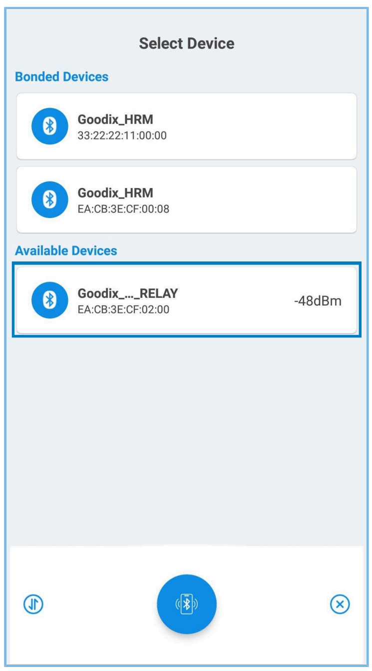

Figure 3-3 Discovering **Goodix_HRS_RSCS_RELAY** on GRToolbox

Note:

If the length of the device name exceeds 14 characters, the middle part of the device name is replaced with an ellipsis.

2. Connect to the HRS RSCS Relay device.

   Select **Goodix_HRS_RSCS_RELAY** to establish connection, and enter the HRS RSCS RELAY interface.

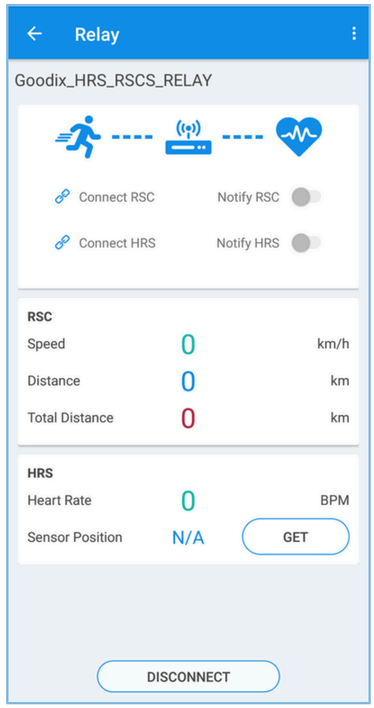

Figure 3-4 HRS RSCS RELAY interface

3. Connect to sensor devices.

   Tap 🔗 to enable the HRS RSCS Relay device to scan and connect to the HRS and RSC devices The interface below is shown after the Relay device is connected to the HRS and RSC devices.

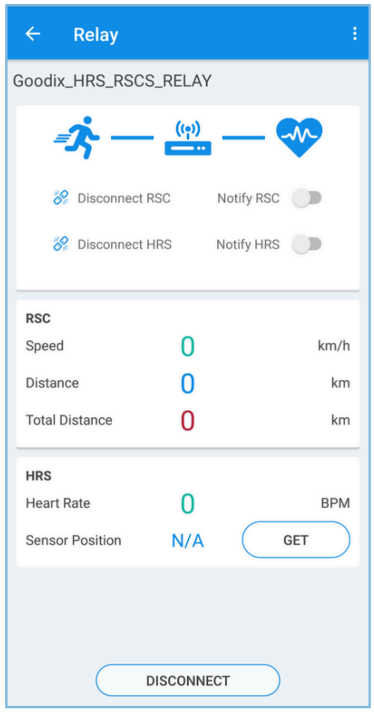

Figure 3-5 Connecting to the HRS and RSC devices

4. Enable sensor notifications.

Tap to enable the HRS RSCS Relay device to notify the HRS and RSC devices to report measurement data. This allows the phone to receive heart rate, running speed, and cadence information relayed from the HRS RSCS Relay device.

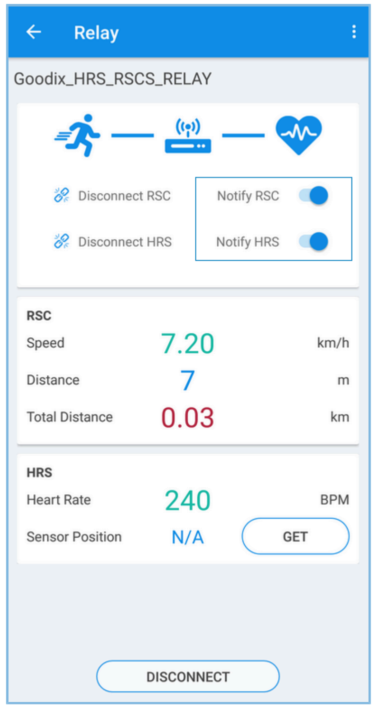

Figure 3-6 Enabling HRS and RSC notifications

5. Read the HRS sensor location.

   Tap **GET** to enable the HRS RSCS Relay device to read the HRS sensor location.

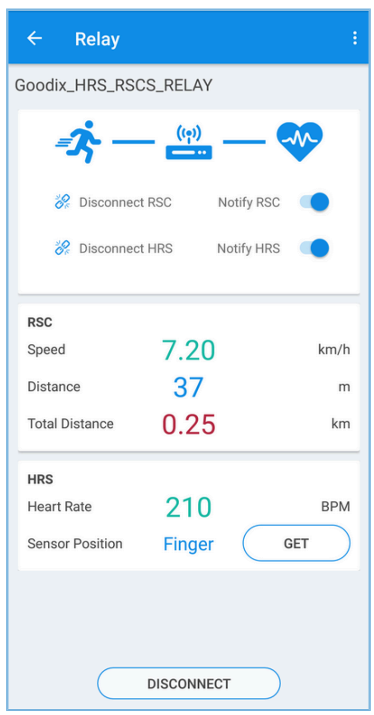

Figure 3-7 Reading the HRS sensor location

If GRToolbox displays information as shown above, the HRS RSCS Relay example runs successfully.

4 Application Details

This chapter introduces the project directory, implementation procedures, and main code of the HRS RSCS Relay example.

4.1 Project Directory

The source code and project file of the HRS RSCS Relay example are in `SDK_Folder\projects\ble\ble_multi_role\ble_app_hrs_rscs_relay`, and project file is in the Keil_5 folder.

Double-click the project file, *ble_app_hrs_rscs_relay.uvprojx*, to view the ble_app_hrs_rscs_relay project directory structure of the HRS RSCS Relay example in Keil. For related files, see Table 4-1.

Table 4-1 File description of ble_app_hrs_rscs_relay

Group	File	Description
gr_profiles	ble_prf_utils.c	This file contains profile-related operational tools.
	hrs.c	This file implements Heart Rate Service.
	hrs_c.c	This file implements Heart Rate Service on the client side.
	rscs.c	This file implements Running Speed and Cadence Service.
	rscs_c.c	This file implements Running Speed and Cadence Service on the client side.
	hrrcps.c	This file implements Heart Rate, Running Speed, and Cadence Service.
user_callback	user_gap_callback.c	This file obtains scanning, connection, and disconnection events.
user_platform	user_periph_setup.c	This file configures App logs, device address, and power management mode.
user_app	main.c	This file contains the main() function.
	user_app.c	This file implements profile registration and logical processing for HRS RSCS Relay applications.

4.2 Implementation Procedures and Code

When the HRS RSCS Relay example starts running, it successively initializes peripherals and BLE Protocol Stack, adds profiles, enables advertising, and waits for connection.

Note:

The main logical code of the HRS RSCS Relay example is in:

- `user_app/user_app.c` in the Keil project directory tree.
- `user_callback/user_gap_callback.c` in the Keil project directory tree.

Implementation procedures of the HRS RSCS Relay example after GRToolbox completes scanning and connection are shown in Figure 4-1:

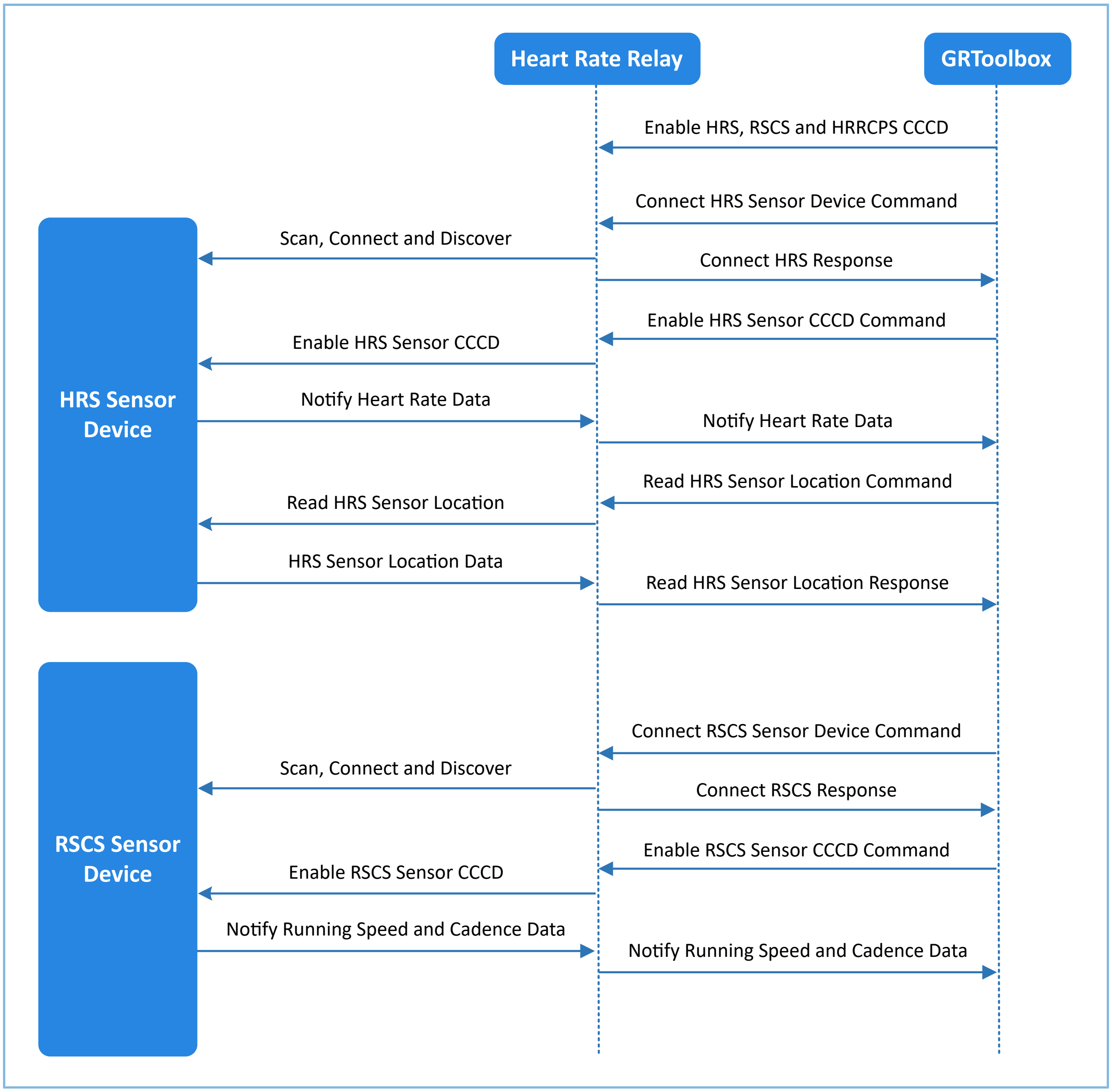

Figure 4-1 Implementation procedures

In the following parts, the HRS device is taken as an example to introduce the interactions between GRToolbox, Relay device, and heart rate sensor, as well as the main code.

- Receive a command from GRToolbox.

  When the HRR Control Point characteristic receives control command data from GRToolbox, it parses the corresponding event and reports to the application layer, and executes the corresponding command. The code snippet is as follows:

```
static void hrrcps_evt_process(hrrcps_evt_t *p_evt)
{
    sdk_err_t   error_code;

    if (p_evt->conn_idx == s_conn_idx_collector)
    {
```

```
        switch (p_evt->evt_type)
        {
            case HRRCPS_EVT_CTRL_PT_IND_ENABLE:
                APP_LOG_DEBUG("HRR Control Point Indication is enabled." );
                break;
            case HRRCPS_EVT_CTRL_PT_IND_DISABLE:
                APP_LOG_DEBUG("HRR Control Point Indication is disabled." );
                break;
            case HRRCPS_EVT_SCAN_HRS:
                if (NO_ACTIVE_STATE ! = g_hrs_active_state)
                {
                    hrrcps_op_error_handler(HRRCPS_CTRL_PT_SCAN_HRS);
                }
                error_code = ble_gap_scan_start();
                if (error_code ! = SDK_SUCCESS)
                {
                    hrrcps_op_error_handler(HRRCPS_CTRL_PT_SCAN_HRS);
                }
                g_hrs_active_state = SCAN_DEV_STATE;
                APP_LOG_DEBUG("Start scanning, target device: HRS." );
                break;
            case HRRCPS_EVT_SCAN_RSCS:
                if (NO_ACTIVE_STATE ! = g_rscs_active_state)
                {
                    hrrcps_op_error_handler(HRRCPS_CTRL_PT_SCAN_RSCS);
                }
                error_code = ble_gap_scan_start();
                if (error_code ! = SDK_SUCCESS)
                {
                    hrrcps_op_error_handler(HRRCPS_CTRL_PT_SCAN_RSCS);
                }
                g_rscs_active_state = SCAN_DEV_STATE;
                APP_LOG_DEBUG("Start scanning, target device: RSCS." );
                break;
             case HRRCPS_EVT_ENABLE_HRS_NTF:
                error_code = hrs_c_heart_rate_meas_notify_set
                                                       (s_conn_idx_hrs_c, true);
                if (error_code ! = SDK_SUCCESS)
                {
                    hrrcps_op_error_handler(HRRCPS_CTRL_PT_HRS_NTF_ENABLE);
                }
                s_user_write_id = USER_WR_HRS_NTF_EN;
                APP_LOG_DEBUG("Enable HRS notification." );
                break;
            case HRRCPS_EVT_DISABLE_HRS_NTF:
                error_code = hrs_c_heart_rate_meas_notify_set
                                                      (s_conn_idx_hrs_c, false);
                if (error_code ! = SDK_SUCCESS)
                {
                    hrrcps_op_error_handler(HRRCPS_CTRL_PT_HRS_NTF_DISABLE);
                }
                s_user_write_id = USER_WR_HRS_NTF_DIS;
                APP_LOG_DEBUG("Disable HRS notification." );
                break;
            case HRRCPS_EVT_ENABLE_RSCS_NTF:
                error_code = rscs_c_rsc_meas_notify_set
                                                      (s_conn_idx_rscs_c, true);
                if (error_code ! = SDK_SUCCESS)
                {
                    hrrcps_op_error_handler(HRRCPS_CTRL_PT_RSCS_NTF_ENABLE);
                }
                s_user_write_id = USER_WR_RSCS_NTF_EN;
```



```
                APP_LOG_DEBUG("Enable RSCS notification." );
                break;
            case HRRCPS_EVT_DISABLE_RSCS_NTF:
                error_code = rscs_c_rsc_meas_notify_set
                                                      (s_conn_idx_rscs_c, false);
                if (error_code ! = SDK_SUCCESS)
                {
                    hrrcps_op_error_handler(HRRCPS_CTRL_PT_RSCS_NTF_DISABLE);
                }
                s_user_write_id = USER_WR_RSCS_NTF_DIS;
                APP_LOG_DEBUG("Disable RSCS notification." );
                break;
            case HRRCPS_EVT_HRS_SENSOR_LOC_READ:
                error_code = hrs_c_sensor_loc_read(s_conn_idx_hrs_c);
                if (error_code ! = SDK_SUCCESS)
                {
                    hrrcps_op_error_handler(HRRCPS_CTRL_PT_HRS_SEN_LOC_READ);
                }
                APP_LOG_DEBUG("Read HRS sensor location." );
                break;
            case HRRCPS_EVT_RSCS_SENSOR_LOC_READ:
                error_code = rscs_c_sensor_loc_read(s_conn_idx_rscs_c);
                if (error_code ! = SDK_SUCCESS)
                {
                    hrrcps_op_error_handler(HRRCPS_CTRL_PT_RSCS_SEN_LOC_READ);
                }
                APP_LOG_DEBUG("Read RSCS sensor location." );
                break;
            default:
                break;
        }
    }
}
```

- The command to connect to the HRS device

  HRRCPS parses the command and reports the HRRCPS_EVT_SCAN_HRS event to the application layer; HRRCPS then starts scanning and searches for the HRS device.

After the target device is discovered, the HRS RSCS Relay device successively stops scanning, gets connected to the HRS device, and discovers Heart Rate Service. The procedures are described as follows.

1. Stop scanning (check whether the device scanned is the HRS device, based on whether the advertising data contains HRS UUID); the code snippet is as follows.

```
void app_adv_report_handler(const uint8_t *p_data, uint16_t length,
                            const gap_bdaddr_t *p_bdaddr)
{
    sdk_err_t error_code;

    if (SCAN_DEV_STATE == g_hrs_active_state)
    {
        if (target_srvc_uuid_find(p_data, length, BLE_ATT_SVC_HEART_RATE))
        {
            memcpy(&s_hrs_target_addr, p_bdaddr, sizeof(gap_bdaddr_t));
            error_code = ble_gap_scan_stop();
            APP_ERROR_CHECK(error_code);
            APP_LOG_DEBUG("Stop scanning for Connecting HRS procedure." );
            g_hrs_active_state = CONN_UNDERWAY_STATE;
            return;
```

```
        }
    }

    if (SCAN_DEV_STATE == g_rscs_active_state)
    {
        If (target_srvc_uuid_find(p_data, length,
            BLE_ATT_SVC_RUNNING_SPEED_CADENCE))
        {
            memcpy(&s_rscs_target_addr, p_bdaddr, sizeof(gap_bdaddr_t));
            error_code = ble_gap_scan_stop();
            APP_ERROR_CHECK(error_code);
            APP_LOG_DEBUG("Stop scanning for Connecting RSCS procedure." );
            g_rscs_active_state = CONN_UNDERWAY_STATE;
            return;
        }
    }
}
```

2. Connect to the HRS device.

```
void app_scan_stop_handler(void)
{
    if (CONN_UNDERWAY_STATE == g_hrs_active_state)
    {
        s_gap_connect_param.peer_addr.addr_type = s_hrs_target_addr.addr_type;
        s_gap_connect_param.peer_addr.gap_addr  = s_hrs_target_addr.gap_addr;
        ble_gap_ext_connect(BLE_GAP_OWN_ADDR_STATIC, &s_gap_connect_param);
    }

    if (CONN_UNDERWAY_STATE == g_rscs_active_state)
    {
        s_gap_connect_param.peer_addr.addr_type = s_rscs_target_addr.addr_type;
        s_gap_connect_param.peer_addr.gap_addr  = s_rscs_target_addr.gap_addr;
        ble_gap_ext_connect(BLE_GAP_OWN_ADDR_STATIC, &s_gap_connect_param);
    }
}
```

3. Discover Heart Rate Service.

```
void app_connected_handler(uint8_t conn_idx,
                           const gap_conn_cmp_t *p_conn_param)
{
    if (GAP_LL_ROLE_MASTER == p_conn_param->ll_role)
    {
        if (CONN_UNDERWAY_STATE == g_hrs_active_state)
        {
            APP_LOG_DEBUG("Connected to HRS, IDX:%d." ,conn_idx);

            s_conn_idx_hrs_c    = conn_idx;
            g_hrs_active_state = CONNECTED_STATE;

            hrs_c_disc_srvc_start(s_conn_idx_hrs_c);
            ble_srv_disc_proc_state_set(HRS_DISC_PROC_ID,
                                        BLE_SRV_DISC_UNDERWAY);

            APP_LOG_DEBUG("Start discovery HRS service." );
        }

        if (CONN_UNDERWAY_STATE == g_rscs_active_state)
        {
```

```
            APP_LOG_DEBUG("Connected to RSCS, IDX:%d." , conn_idx);

            s_conn_idx_rscs_c   = conn_idx;
            g_rscs_active_state = CONNECTED_STATE;

            rscs_c_disc_srvc_start(s_conn_idx_rscs_c);;
            ble_srv_disc_proc_state_set(RSCS_DISC_PROC_ID,
                                        BLE_SRV_DISC_UNDERWAY);

            APP_LOG_DEBUG("Start discovery RSCS service." );
        }
    }

    if (GAP_LL_ROLE_SLAVE == p_conn_param->ll_role)
    {
        APP_LOG_DEBUG("Connected to Collector, IDX:%d." , conn_idx);
        s_conn_idx_collector = conn_idx;
    }
}
```

- The command to enable HRS notification

  HRRCPS parses the command and reports the HRRCPS_EVT_ENABLE_HRS_NTF event to the application layer; HRRCPS then enables HRS notification and relays the received heart rate data to GRToolbox.

```
static void hrs_c_evt_process(hrs_c_evt_t *p_evt)
{
    ...
    switch (p_evt->evt_type)
    {
        ...
        case HRS_C_EVT_HR_MEAS_VAL_RECEIVE:
            for (rr_intervals_idx = 0; rr_intervals_idx <
               p_evt->value.hr_meas_buff.rr_intervals_num; rr_intervals_idx++ )
            {
             hrs_rr_interval_add(
                     p_evt->value.hr_meas_buff.rr_intervals[rr_intervals_idx]);
            }
            hrs_sensor_contact_detected_update
                        (p_evt->value.hr_meas_buff.is_sensor_contact_detected);

            hrs_heart_rate_measurement_send(s_conn_idx_collector,
                                            p_evt->value.hr_meas_buff.hr_value,
                                    p_evt->value.hr_meas_buff.energy_expended);
            break;
        ...
        default:
            break;
    }
}
```

- The command to obtain the HRS sensor location

  HRRCPS parses the command and reports the HRRCPS_EVT_HRS_SENSOR_LOC_READ event to the application layer in BLE Protocol Stack; HRRCPS then reads the HRS sensor location and relays the data obtained to GRToolbox.

```
static void hrs_c_evt_process(hrs_c_evt_t *p_evt)
{
```



```
    ...
    switch (p_evt->evt_type)
    {
        ...
        case HRS_C_EVT_SENSOR_LOC_READ_RSP:
            APP_LOG_DEBUG("HRS sensor location is got." );
            hrs_sensor_location_set((hrs_sensor_loc_t)p_evt->value.sensor_loc);
            rsp_val.cmd_id       = HRRCPS_CTRL_PT_HRS_SEN_LOC_READ;
            rsp_val.rsp_id       = HRRCPS_RSP_ID_OK;
            rsp_val.is_inc_prama = true;
            rsp_val.rsp_param    = p_evt->value.sensor_loc;
            error_code = hrrcps_ctrl_pt_rsp_send(s_conn_idx_collector,
                                                 &rsp_val);
            APP_ERROR_CHECK(error_code);
            break;
        default:
            break;
    }
}
```

Note:

You can use GRToolbox to control the interactions between the HRS RSCS Relay device and the RSCS device, which are similar to the procedures mentioned above, and therefore are not explained in this document.