GR5xx HID Mouse Example Application

Version: 3.0

Release Date: 2023-03-30

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: Floor 12-13, Phase B, Tengfei Industrial Building, Futian Free Trade Zone, Shenzhen, China

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

Website: www.goodix.com

Preface

Purpose

This document introduces how to run Human Input Device (HID) Service and Bluetooth Low Energy (Bluetooth LE) GR5xx HID mouse example for the first time and its application details, to help users quickly get started with secondary development.

Audience

This document is intended for:

- GR5xx user
- GR5xx developer
- GR5xx tester
- GR5xx technical support engineer
- Technical writer

Release Notes

This document is the second release of *GR5xx HID Mouse Example Application*, corresponding to Bluetooth LE GR5xx System-on-Chip (SoC) series.

Revision History

Version	Date	Description
1.0	2023-01-10	Initial release
3.0	2023-03-30	Updated descriptions about GR5xx SoCs.

Contents

1 Introduction

The GR5xx Human Input Device (HID) Mouse example implements an HID firmware example.

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

Table 1-1 Reference documents

Name	Description
Developer guide of the specific GR5xx SoC	Introduces GR5xx Software Development Kit (SDK) and how to develop and debug applications based on the SDK.
J-Link/J-Trace User Guide	Provides J-Link operational instructions. Available at https://www.segger.com/downloads/jlink/UM08001_JLink.pdf.
Keil User Guide	Offers detailed Keil operational instructions. Available at https://www.keil.com/support/man/docs/uv4/.
Bluetooth Core Spec	Offers official Bluetooth standards and core specification from Bluetooth SIG.
Bluetooth GATT Spec	Provides details about Bluetooth profiles and services. Available at https://www.bluetooth.com/specifications/gatt.
GProgrammer User Manual	Lists GProgrammer operational instructions including downloading firmware to and encrypting firmware on GR5xx System-on-Chips (SoCs).

2 HID over GATT Profile (HOGP)

This chapter introduces the device roles, HID Service, and security requirements defined by HOGP.

2.1 Device Roles

HOGP defines two roles: HID Device and HID Host.

- HID Device

  The HID Device shall perform the GAP Peripheral role as a GATT Server. Common HID Devices include keyboards and mice.

  An HID Device shall contain at least an HID Service instance, a Battery Service (BAS) instance, a Device Information Service (DIS) instance, and optionally a Scan Parameters Service instance. An HID Device can contain one or more other types of GATT Service instances that do not serve as parts of HOGP.

  The ble_app_hids_mouse example used to implement the HID Device in the GR5xx SDK contains an HID Service instance, a BAS instance, and a DIS instance.

- HID Host

  The HID Host, parsing the input data delivered by the HID Device, shall perform the GAP Central role as a GATT Client. Common HID Host examples are Android phones. The HID Host is responsible for scanning, connecting to, and configuring the HID Device. When the connection between the HID Device and HID Host is established, the HID Host can receive and read data from as well as write data to the HID Device.

2.2 HID Service

The HID Service presents data and associated formats of the HID Device (defined in *USB HID Specification*) to the HID Host.

The HID Service uses characteristics to access data on an HID Device. For characteristic details, see Table 2-1.

Table 2-1 HID service characteristics

Characteristic		UUID	Type	Support	Security	Properities
Protocol Mode		2A4E	16 bits	Mandatory for Boot Protocol Mode support	None	Read, Write
Report	Input Report Type	2A4D	16 bits	Mandatory to support at least one Report Type if the Report characteristic is supported	None	Read, Notify, Write
	Output Report Type					Read, Write, Write Without Response
	Feature Report Type					Read, Write
Report Map		2A4B	16 bits	Mandatory	None	Read

Characteristic	UUID	Type	Support	Security	Properities
Boot Keyboard Input Report	2A22	16 bits	Mandatory for keyboards	None	Read, Notify, Write
Boot Keyboard Output Report	2A32	16 bits	Mandatory for keyboards	None	Read, Write, Write Without Response
Boot Mouse Input Report	2A33	16 bits	Mandatory for mice	None	Read, Notify, Write
HID Information	2A4A	16 bits	Mandatory	None	Read
HID Control Point	2A4C	16 bits	Mandatory	None	Write Without Response

- Protocol Mode Characteristic: Used to expose the current protocol mode, or set the desired protocol mode.
- Report Characteristic: Used to exchange data between HID Device and HID Host.
- Report Map Characteristic: Used to define formatting information for the data transferred between HID Device and HID Host.
- Boot Keyboard Input Report/Boot Keyboard Output Report Characteristic: Used to enable an HID Host (running in Boot Protocol Mode) to transmit Input Report or Output Report data in a fixed format and at a fixed length to an HID Device corresponding to the Boot Keyboard.
- Boot Mouse Input Report Characteristic: Used to enable an HID Host (running in Boot Protocol Mode) to transmit Input Report data in a fixed format and at a fixed length to an HID Device corresponding to the Boot Mouse
- HID Information Characteristic: Used to hold a set of values known as the HID Device's HID Attributes.
- HID Control Point Characteristic: A control-point attribute, used to define the HID Command to suspend or exit supending.

2.3 Security Requirements

According to *Bluetooth Core Spec*, LE Security Mode 1 includes Security Level 2 and Security Level 3.

- Security Level 2: Encrypted Link required; MITM protection not necessary.
- Security Level 3: MITM-protected encrypted link required.

According to *HOGP Specification*, the HID Device shall support either Security Level 2 or 3.

- The Security Property of all characteristics supported by the HID Service shall be set to Security Mode 1 and either Security Level 2 or 3.
- It is recommended that all characteristics specified by Device Information Service, Scan Parameters Service, and BAS should be set to the same LE Security Mode and Security Level.

Users can set the security parameters for a GR5xx HID mouse example by using gap_params_init(). For details, see "Section 4.2.2 Configuring Security Parameters".

3 Initial Operation

This chapter introduces how to run and verify the HID mouse example in the GR5xx SDK.

Note:

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

3.1 Preparation

Perform the following tasks before running an HID mouse example.

- **Hardware preparation**

Table 3-1 Hardware preparation

Name	Description
Development board	Starter Kit Board (SK Board) of the corresponding SoC
Android phone	A mobile phone running on Android 5.0 (KitKat) and later
Connection cable	USB Type C cable (Micro USB 2.0 cable for GR551x SoCs)

- **Software preparation**

Table 3-2 Software preparation

Name	Description
Windows	Windows 7/Windows 10
J-Link driver	A J-Link driver. Available at https://www.segger.com/downloads/jlink/.
Keil MDK5	An integrated development environment (IDE). MDK-ARM Version 5.20 or later is required. Available at https://www.keil.com/download/product/.
GProgrammer (Windows)	A programming tool. Available in SDK_Folder\tools\GProgrammer.
GRUart (Windows)	A serial port debugging tool. Available in SDK_Folder\tools\GRUart.

3.2 Firmware Programming

The source code of the HID mouse example is in SDK_Folder\projects\ble\ble_peripheral\ble_app_hids_mouse.

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

Note:

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

3.3 Test and Verification

After the SK Board, Android phone, and GRUart are ready, you can test and verify an HID mouse example.

1. Press **RESET** on the Board, and the Board enters Advertising mode.
2. Open the Bluetooth setting interface on the phone, and turn **Bluetooth** on. Wait until the phone discovers **Goodix_Mouse**.
3. Tap **Goodix_Mouse** to connect it to the phone.
4. Enter **123456** into **Pin Code** in the pop-up dialog box.

After pairing, you can see the device named **Goodix_Mouse** under **Paired devices** on the phone, and the device shows as **Connected**. As shown in Figure 3-1, you can enter **up**, **down**, **left**, and **right** on GRUart to move the mouse arrow. In this case, in the **Tx** area on GRUart, clear **Hex** and **NewLine** for **Format**.

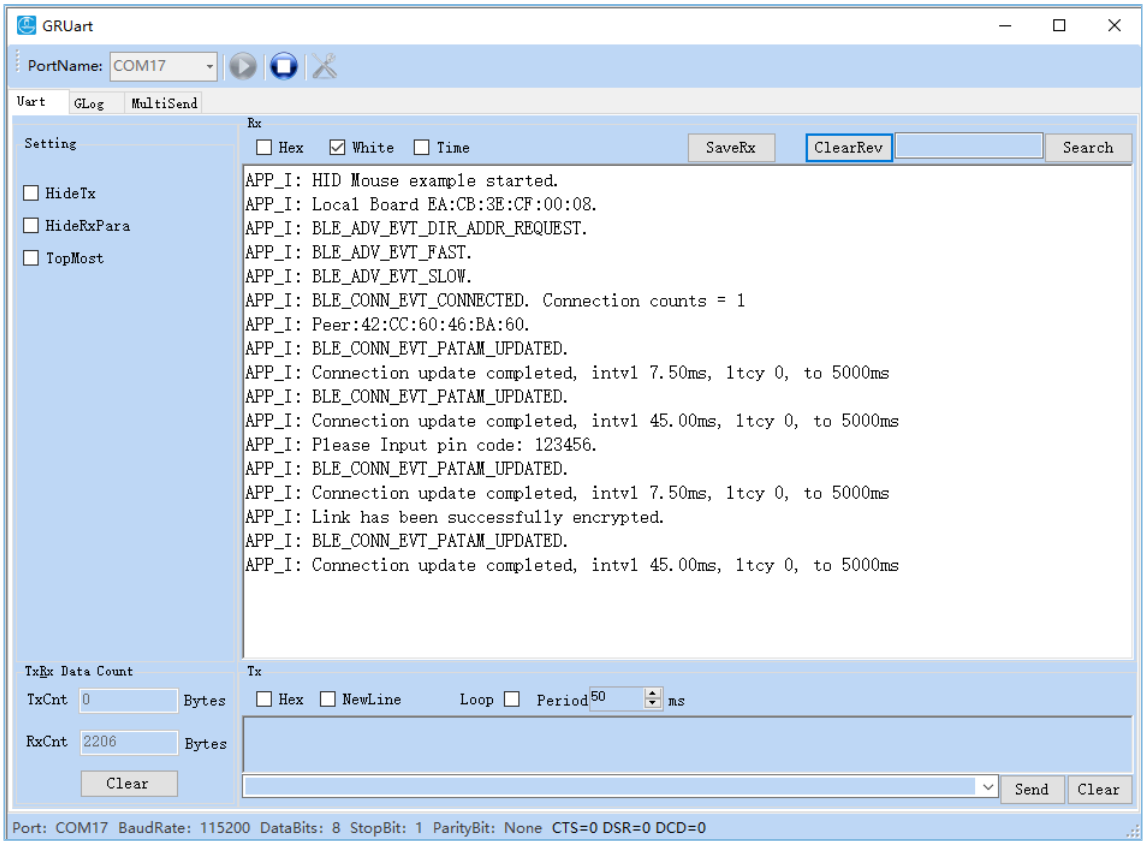

Figure 3-1 Mouse arrow move command on GRUart

You can see the moves of the mouse arrow in Figure 3-2.

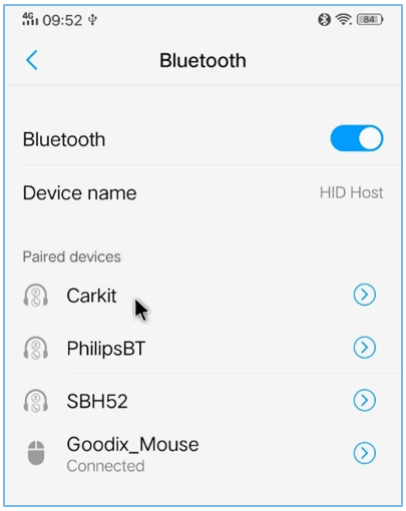

Figure 3-2 GR5xx mouse arrow on an Android phone

In addition, you can use the HID mouse example in media play control scenarios. Enter and send **up+** or **down+** on GRUart to increase or decrease the volume respectively. Enter and send **ok+** to stop or resume playing. Enter and send **right+** to switch to the next track, and use **left+** to switch to the previous track. It should be noted that media play control functions of the HID mouse example may be unavailable in some scenarios due to version limitations on Android operating systems.

4 Application Details

This chapter introduces the running procedures and major code of the HID mouse example.

4.1 Running Procedures

The running procedures of a GR5xx HID mouse example can be divided into two phases: boot and interactive processing. The following figure displays the procedures specific to phase.

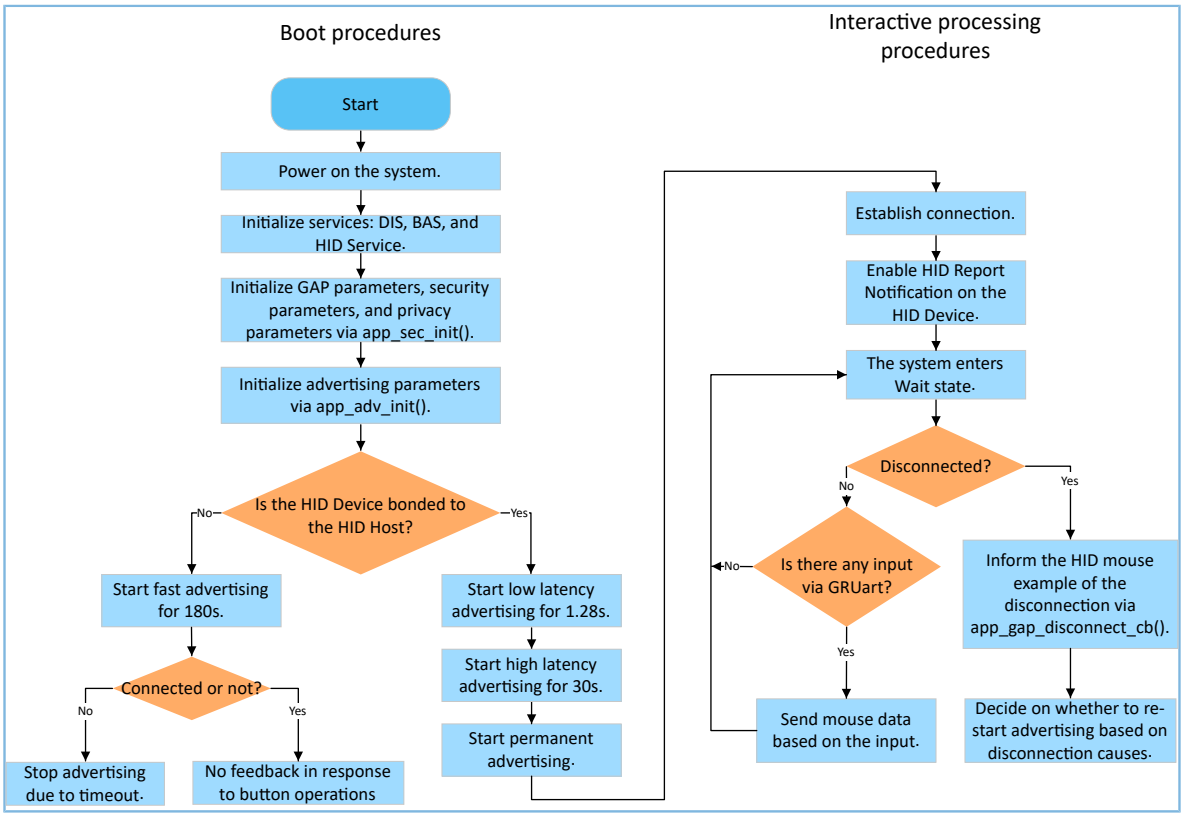

Figure 4-1 Running procedures of a GR5xx HID mouse example

During booting, the relation (bond and unbonded) between HID Device and HID Host affects the advertising parameters. For details, see “Connection Establishment” in *HOGP Specification*.

Note:

If the input parameter of void adv_sec_init(bool erase_bond) is true, the bond information is erased after each reboot.

4.2 Major Code

The major code of the HID mouse example is listed in the following sections under the Keil project directory.

4.2.1 Initializing HID Service

Configure initialization parameters of HID Service by using the hids_init() function. The initialization parameters contain rep_map_data. According to formats specified in *USB HID Specification*, the rep_map_data array defines Report Map characteristics of mouse report and media key report.

Path: `user_app\user_mouse.c` under the project directory

Name: hids_init();

```
static void hids_init(void)
{
    hids_init_t hids_init;

    hids_init.report_map.p_map = (uint8_t*)&rep_map_data;
    hids_init.report_map.len   = sizeof(rep_map_data);

    …

    hids_service_init(&hids_init);
}
```

Note:

The data length of static uint8_t rep_map_data should not excceed the REPORT_MAP_MAX_SIZE defined in *hids.h*.

4.2.2 Configuring Security Parameters

The app_sec_init() function sets the following security parameters in compliance with security requirements specified in “Security Requirements” according to *HOGP Specification*.

The app_sec_init() function enables privacy mode by using ble_gap_privacy_params_set(). In privacy mode, the HID mouse example generates a device address at an interval of the value of PRIVACY_RENEW_DURATION and does not respond to Connect requests from the bonded HID Host by using Public Address. You can define the value of PRIVACY_RENEW_DURATION in *user_app.c*.

Path: `user_app\user_app.c` under the project directory

Name: app_sec_init();

```
static void app_sec_init(bool erase_bond)
{
    …
    error_code = ble_gap_privacy_params_set(PRIVACY_RENEW_DURATION, true);
    APP_ERROR_CHECK(error_code);

    //set the default security parameters.
    sec_param_t sec_param =
    {
        .level   = SEC_MODE1_LEVEL3,
        .io_cap  = IO_DISPLAY_ONLY,
        .oob  = false,
        .auth  = AUTH_BOND | AUTH_MITM | AUTH_SEC_CON,
        .key_size  = 16,
        .ikey_dist = KDIST_ALL,
        .rkey_dist = KDIST_ALL,
    };
    error_code = ble_sec_params_set(&sec_param);
    APP_ERROR_CHECK(error_code);
```

```
}
```

The app_sec_rcv_enc_req_handler() function in *user_app.c* is used to respond to encrypted pairing requests from the HID Host. In "Section 3.3 Test and Verification", the input pin code is the tk value set in the app_sec_rcv_enc_req_cb() function.

Path: `user_callbacks\user_sm_callback.c` under the project directory

Name: app_sec_rcv_enc_req_cb();

```
static void app_sec_rcv_enc_req_handler(uint8_t conn_idx, const ble_sec_evt_enc_req_t
                                        *p_enc_req)
{
    …
    switch (p_enc_req->req_type)
    {
        …
        case TK_REQ:
            APP_LOG_INFO("Please Input pin code: 123456");
            cfm_enc.req_type = TK_REQ;
            cfm_enc.accept   = true;
            tk = 123456;
            memset(cfm_enc.data.tk.key, 0, 16);
            cfm_enc.data.tk.key[0] = (uint8_t)((tk & 0x000000FF) >> 0);
            cfm_enc.data.tk.key[1] = (uint8_t)((tk & 0x0000FF00) >> 8);
            cfm_enc.data.tk.key[2] = (uint8_t)((tk & 0x00FF0000) >> 16);
            cfm_enc.data.tk.key[3] = (uint8_t)((tk & 0xFF000000) >> 24);
            break;
    }
    ble_sec_enc_cfm(conn_idx, &cfm_enc);
}
```

4.2.3 Sending UART Instructions

Users can send control instructions to the HID through GRUart and perform operations by pressing buttons on the Board. When the app_key_evt_handler() function of the HID mouse example receives instructions delivered by GRUart, the function calls hids_input_rep_send() in the HIDS module to transmit the mouse data to the HID Host. The mouse data can be divided into two types: mouse_data_t and media_data_t.

Path: `gr_profiles\hids.c` under the project directory

Name: hids_input_rep_send();

```
sdk_err_t hids_input_rep_send(uint8_t conn_idx, uint8_t rep_idx,
                                        uint8_t *p_data, uint16_t length)
{
    static const uint8_t char_idx[] = {HIDS_IDX_INPUT1_REPORT_VAL,
                                           HIDS_IDX_INPUT2_REPORT_VAL,
 HIDS_IDX_INPUT3_REPORT_VAL};
    sdk_err_t   error_code = SDK_ERR_NTF_DISABLED;
    if(rep_idx >= IN_REPORT_MAX_COUNT || p_data == NULL || length == 0)
    {
        return SDK_ERR_INVALID_PARAM;
    }
    length = ((length > HIDS_REPORT_MAX_SIZE) ? HIDS_REPORT_MAX_SIZE : length);
    memcpy(&s_hids_env.input_report_val[rep_idx], p_data, length);
    if(s_hids_env.input_cccd[rep_idx][conn_idx] == PRF_CLI_START_NTF)
    {
        error_code = hids_in_rep_notify(conn_idx, char_idx[rep_idx], p_data, length);
```

```
    }
    return error_code;
}
```

Path: `user_app\use_mouse.h` under the project directory

Name: mouse_data_t and media_data_t;

```
typedef struct
{
    bool left_button_press;
    bool middle_button_press;
    bool right_button_press;
    int8_t x_delta;
    int8_t y_delta;
    int8_t wheel_delta;
} mouse_data_t;

typedef struct
{
    uint8_t play_pause:1;
    uint8_t al_control:1;
    uint8_t next_track:1;
    uint8_t previous_track:1;
    uint8_t volume_down:1;
    uint8_t volume_up:1;
    uint8_t ac_foward:1;
    uint8_t ac_back:1;
} media_data_t;
```

4.2.4 Disconnecting from HID Host

When the HID Device is disconnected from the HID Host, the Blueooth LE Protocol Stack notifies the disconnection event to the HID mouse example by using gap_cb_fun_t::app_gap_disconnect_cb(). The ble_adv_disconnected() function decides on whether to restart advertising based on the disconnection reason.

According to *HOGP Specification*, the HID Device should restart advertising if the connection is terminated due to link loss. To simplify tests, the HID mouse example restarts advertising when the disconnection reason is Remote User Terminated Connection. If the HID Device is bonded to the HID Host, the HID mouse example enters High Latency Advertising, Low Latency Advertising, and Permanent Advertising successively.

Path: `ble_module\ble_advertising.c` under the project directory

Name: ble_adv_disconnected();

```
static void ble_adv_disconnected(void)
{
    if (adv_env.adv_mode_cfg.adv_on_disconnect_enabled && !adv_env.adv_act_exist)
    {
        ble_advertising_start(BLE_ADV_MODE_DIRECTED_HIGH_DUTY);
    }
}
```