GR55xx HID Mouse Example Application

Version: 1.7

Release Date: 2021-04-20

Shenzhen Goodix Technology Co., Ltd.

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 run Human Input Device (HID) Service and GR55xx 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:

- GR55xx user
- GR55xx developer
- GR55xx tester
- GR55xx technical support engineer
- Technical writer

Release Notes

This document is the fifth release of *GR55xx HID Mouse Example Application*, corresponding to GR55xx 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	Modified the booting procedures of a GR55xx HID mouse example in “Boot”.
1.6	2020-06-30	Updated the document version based on SDK changes.
1.7	2021-04-20	Optimized descriptions in “Initial Operation” and “Application Details”.

Contents

1 Introduction

The GR55xx 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 GR55xx SoC	Introduces the software/hardware and quick start guide of the specific GR55xx SoC in use.
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.
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 GR55xx 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 GR55xx 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 details about the characteristics, 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 GR55xx 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 GR55xx HID mouse example in the GR55xx SDK.

Note:

SDK_Folder is the root directory for the GR55xx SDK in use.

3.1 Supported Development Platform

You can use and modify the HID mouse example on the following platform.

Table 3-1 Supported development platform

Hardware Platform	Development Board Model
GR551x development kit	GR5515-SK-BASIC

3.2 Firmware Download

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

You can download *ble_app_hids_mouse_fw.bin* to the GR55xx Starter Kit Board (GR55xx SK Board) through GProgrammer. For details, see *GProgrammer User Manual*.

Note:

- The *ble_app_hids_mouse_fw.bin* is in `SDK_Folder\projects\ble\ble_peripheral\ble_app_hids_mouse\build\`.
- You can find GProgrammer in `SDK_Folder\tools\GProgrammer`.

3.3 Test and Verification

The hardware required for test and verification is listed in Table 3-2.

Table 3-2 Hardware resource

Name	Description
Android device	Phones running on Android 5.0 (KitKat) or later versions

You can use an Android phone to test and verify an HID mouse example.

1. Press **RESET** on the GR55xx SK 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 long press the **UP**, **DOWN**, **LEFT**, or **RIGHT** button on the GR55xx SK Board to move the mouse arrow.

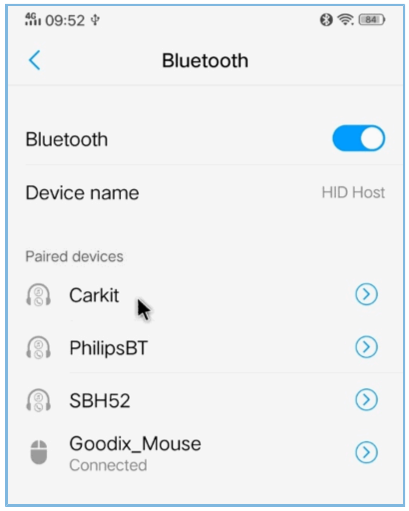

Figure 3-1 GR55xx mouse arrow on an Android phone

In addition, you can use the HID mouse example in media play control scenarios. Press **UP** or **DOWN** twice on the GR55xx SK Board to increase or decrease the volume respectively. Press **OK** twice to stop or resume playing. Press **RIGHT** twice to switch to the next track, and press **LEFT** twice 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 GR55xx HID mouse example.

4.1 Running Procedures

The running procedures of a GR55xx HID mouse example can be divided into two phases: boot and interactive processing. The following figure displays the procedures specific to phase.

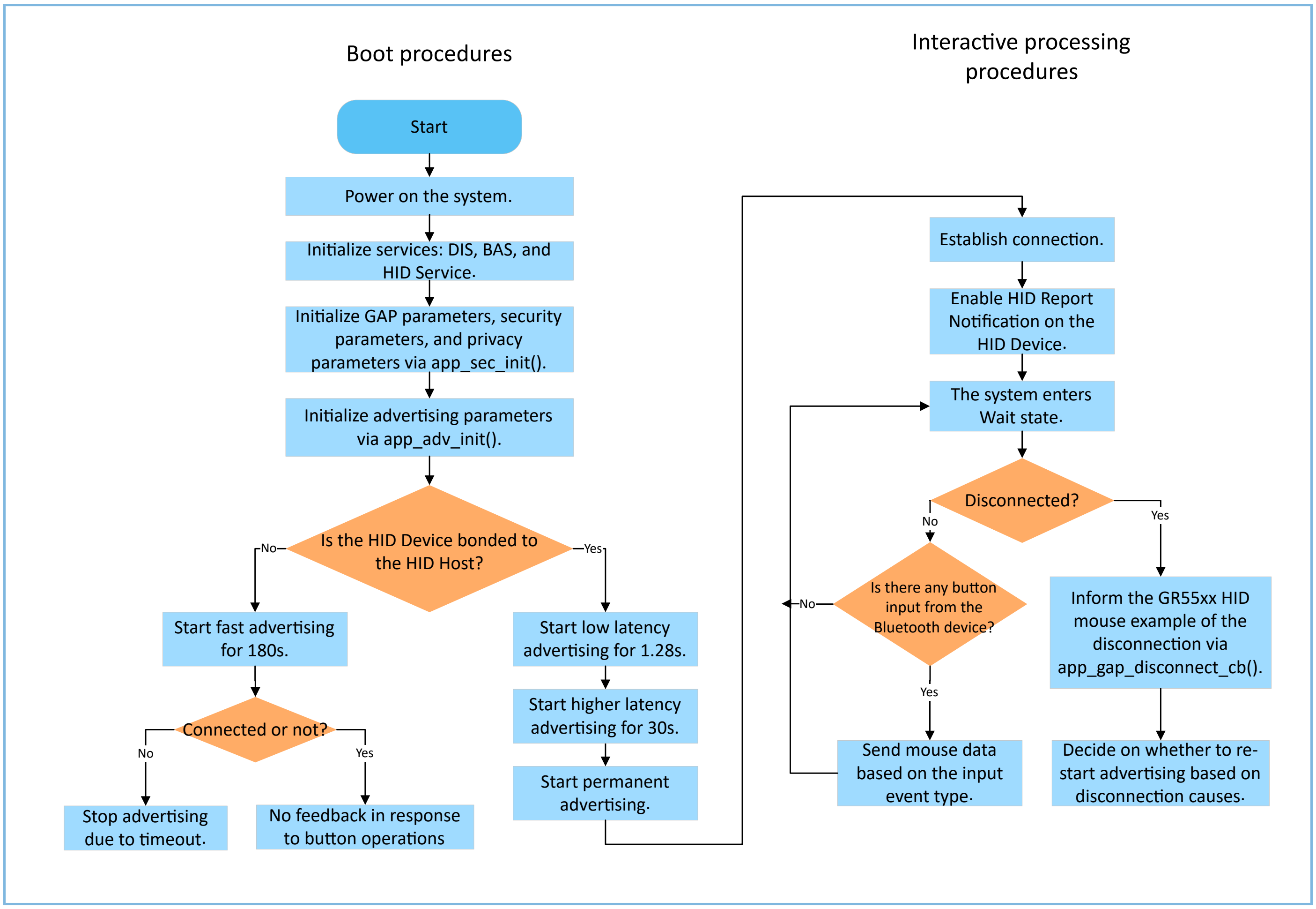

Figure 4-1 Running procedures of a GR55xx HID mouse example

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 value of bool erase_bond in 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 GR55xx 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 be equal to or shorter than 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 "Section 2.3 Security Requirements" according to *HOGP Specification*. For details about sec_param_t setting, see "Security Manager (SM)" in *GR55xx Bluetooth Low Energy Stack User Guide*.

The app_sec_init() function enables privacy mode by using ble_gap_privacy_params_set(). In privacy mode, the GR55xx 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_cb() function in *user_sm_callback.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_cb(uint8_t conn_idx, sec_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);
}
```

For more information about how to handle pairing and encryption requests, refer to “Enable Bonding” in *GR55xx Bluetooth Low Energy Stack User Guide*.

4.2.3 Sending Button Requests

When users perform operations by pressing buttons on the GR55xx SK Board, the app_key_evt_handler() function of the GR55xx HID mouse example receives the button-related events from the Board Support Package (BSP) layer and 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 BLE 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 GR55xx 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 GR55xx HID mouse example enters Low Latency Advertising, Higher 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);
    }
```



```
}
```