AN0145
应用笔记

AT32 HOGP开发指南

前言

人机接口设备(Human Interface Devices, HID)的主要目的是基于 USB 实现者论坛所维护的规格书，提供用户与计算器之间的接口。HID 涵盖了显示器、键盘、网络摄影机和鼠标甚至到 VR 耳机、游戏手把和触控屏幕的各种装置。HID 传统上通过 USB 接口连接到计算器，然而，HID 设备也可以通过使用无线技术与计算器连接。通过使用 HID 规范，可以开发出符合通用标准的计算器外设，从而与大多数装置兼容。

支持型号列表：

支持型号	AT32WB415

目录

1 HOGP 概述 5

1.1 HID 角色 5

1.1.1 HID 主机 5

1.1.2 HID 设备 5

1.2 Bluetooth Profile 5

2 例 HID 装置实做 6

2.1 报告映射讨论 6

2.2 修改装置类型 7

2.3 资源准备 8

2.4 软件设计 8

2.5 实验效果 17

2.6 按键定义 18

2.7 HOGP 相关 AT command 19

3 文档版本历史 21

表目录

表 1. HID 装置的必须服务 5
表 2. HOGP 相关 AT command 19
表 3. 文档版本历史 21

图目录

图 1. Appearance Part of ADV Data..... 7
图 2. GAP Client Gets Device Apperance 8
图 3. GAP Client Gets Device Apperance 8

1 HOGP 概述

HOGP(HID over GATT)是由 Bluetooth SIG 维护的蓝牙配置文件规范，通过低功耗蓝牙实现 HID 的配置文件并作为与计算器之间的接口，消除 HID 中对电线或物理连接的需求。

1.1 HID 角色

在 HID 中，以及在 HOGP 的延伸中，定义了两个角色，这些角色是 HID 主机和 HID 设备。HID 设备一次只能主动连接到单个 HID 主机；然而一个 HID 主机可以连接到多个 HID 设备。有关主机和设备如何交互的细节，请参阅 HOGP 规范中的第 3 章 HID 设备需求和第 4 章 HID 主机需求和行为。

1.1.1 HID 主机

HID 主机实现蓝牙 GAP 中的中心(Central)角色，接收输入讯息并使用提供的数据进行更进一步的处理。例如，计算器会是 HID 主机，因为这是接收 HID 数据的设备。HID 主机对 HID 功能的正确执行有许多责任与需求，但在此开发指南中所提供的范例仅实现了 HID 设备，因此本文檔不涉及这些内容。想要了解 HID 主机的职责和需求，请参阅 HOGP 规范中的第 2 章配置、第 4 章 HID 主机需求和行为、第 5 章连线建立。

1.1.2 HID 设备

HID 设备实现蓝牙 GAP 中的外设(Peripheral)角色，负责将输入信息传输到中央计算器。例如，鼠标或键盘会是 HID 设备，因为它们将输入数据发送到计算器设备。

1.2 Bluetooth Profile

要透过蓝牙实现 HID 功能，那就必须要事先定义好规范，如此主机及装置双方才能够顺利通信，由于本开发指南是针对装置的部份进行说明，故此处只列出装置所需的服务。

表 1. HID 装置的必须服务

Service	Requirement
HID Service	Mandatory
Battery Service	Mandatory
Device Information Service	Mandatory
Scan Parameters Service	Optional

这边要注意到 Device Information Service 中的 PnP 特征，如果是开发一个标准的键盘或鼠标，请根据已经注册的 Vendor ID 及 Product ID 填入其中。

2 例 HID 装置实做

HOGP 工程是一个在 AT32WB415 上实现 HOGP 规范的低功耗蓝牙工程，该工程使用 HOGP 规范指定的蓝牙外设角色，演示了 HOGP 的键盘及鼠标。通过使用通知(Notification)和 GATT 配置文件，该项目能够通过低功耗蓝牙 HID 装置并与 HID 主机正确连接。HOGP 工程中有几个复杂的部份，在本文档中进行了讨论，这些部份组合在一起以实现整体功能。

2.1 报告映射讨论

HOGP 工程依赖于报告映像，报告映射是 HOGP 用于实现低功耗蓝牙 HID 功能的结构。报告映像用于告诉主机许要从 HID 设备获得什么功能和什么样的数据，在原始的 HID USB 规范中，报告映射称为报告描述符，报告映射和报告描述符是相同的，它们之间唯一的区别是报告映射用于 HOGP, 报告描述符用于 HID USB。

该工程使用的报告映像可以在 app_hid.c 文件中找到，并且包含在 gHIDReportDescriptor 变量中，此工程中的报告映像包含键盘的输入信息。该结构分为多个部份，应用程序以及 HID 主机能够通过使用报告 ID 字段找到所需的部份。可以轻松修改报告映像以包含不同的输入或修改已经存在的输入，

USB 实现者论坛的设备工作组有一个报告描述符建构器工具，可以用于轻松创建报告映射。下面的程序代码显示了报告映像结构片段，所有字段都有标签，可以轻松修改。例如，如果需要不同数量的键盘按键，则可以轻松修改相关字段。

- HID Report

```
0x05, 0x01,                    // Usage Page (Generic Desktop Ctrls)
0x09, 0x06,                    // Usage (Keyboard)
0xA1, 0x01,                    // Collection (Application)
0x85, HIDS_KB_REPORT_ID,       //   Report ID (13)
0x05, 0x07,                    //   Usage Page (Kbrd/Keypad)
0x19, 0xE0,                    //   Usage Minimum (0xE0)
0x29, 0xE7,                    //   Usage Maximum (0xE7)
0x15, 0x00,                    //   Logical Minimum (0)
0x25, 0x01,                    //   Logical Maximum (1)
0x75, 0x01,                    //   Report Size (1)
0x95, 0x08,                    //   Report Count (8)
0x81, 0x02,                    //   Input (Data,Var,Abs,No Wrap,Linear,Preferred State,No Null Position)
0x95, 0x01,                    //   Report Count (1)
0x75, 0x08,                    //   Report Size (8)
0x81, 0x01,                    //   Input (Const,Array,Abs,No Wrap,Linear,Preferred State,No Null Position)
0x95, 0x05,                    //   Report Count (5)
0x75, 0x01,                    //   Report Size (1)
0x05, 0x08,                    //   Usage Page (LEDs)
0x19, 0x01,                    //   Usage Minimum (Num Lock)
0x29, 0x05,                    //   Usage Maximum (Kana)
0x91, 0x02,                    //   Output (Data,Var,Abs,No Wrap,Linear,Preferred State,No Null
```

```
Position,Non-volatile)
0x95, 0x01,                    //  Report Count (1)
0x75, 0x03,                    //  Report Size (3)
0x91, 0x01,                    //  Output (Const,Array,Abs,No Wrap,Linear,Preferred State,No Null
Position,Non-volatile)
0x95, 0x06,                    //  Report Count (6)
0x75, 0x08,                    //  Report Size (8)
0x15, 0x00,                    //  Logical Minimum (0)
0x25, 0xFF,                    //  Logical Maximum (-1)
0x05, 0x07,                    //  Usage Page (Kbrd/Keypad)
0x19, 0x00,                    //  Usage Minimum (0x00)
0x29, 0xFF,                    //  Usage Maximum (0xFF)
0x81, 0x00,                    //  Input (Data,Array,Abs,No Wrap,Linear,Preferred State,No Null Position)
0xC0,                          // End Collection
```

2.2 修改装置类型

只有当开发者正确地填入装置类型参数，主机端才能显示出合适的装置图标，即便在广播的时候能够正确辨识出装置的类型，但如果没有修改 GAP 中的 Appearance 数值，联机后的装置图标依然会显示异常，以下特别列出在开发时容易疏忽的地方：

1. 广播时的 Appearance，主机扫描到时就会将装置类型显示出来，比如说键盘或是鼠标，在代码中位于 user_config.h 里面的宏定义 APP_HID_ADV_DATA_APPEARANCE, 0x03C2 代表鼠标, 0x03C1 代表键盘，更详细的内容可以参考 SIG 的 Spec.

图 1. Appearance Part of ADV Data

```
/**
 * Appearance part of ADV Data
 * ------------------------------------------------------------------------------
 * x03 - Length
 * x19 - Appearance
 * x03\x00 - Generic Thermometer
 *   or
 * xC2\x03 - HID Mouse
 * xC1\x03 - HID Keyboard
 * xC3\x03 - HID Joystick
 * ------------------------------------------------------------------------------
 */
//广播数据显示设置
#define APP_HID_ADV_DATA_APPEARANCE      "\x03\x19\xC2\x03"
#define APP_ADV_DATA_APPEARANCE_LEN      (4)
```

2. 联机建立后，主机会来读取装置的 GAP, 这个时候就会得到这个装置是属于哪一种类型，如果没有正确填写，在主机端的驱动安装完成后，显示的装置类型会与预期的不同。这个部份的代码位于 app_task.c 中的 gapc_get_dev_info_req_ind_handler 函式，代码中已将键盘及鼠标的 Appearance 给定义起来，使用者可以依据需求扩充。

图 2. GAP Client Gets Device Apperance

```
311   case GAPC_DEV_APPEARANCE:
312   {
313     // Allocate message
314     struct gapc_get_dev_info_cfm *cfm = KE_MSG_ALLOC(GAPC_GET_DEV_INFO_CFM,
315                                                      src_id, dest_id,
316                                                      gapc_get_dev_info_cfm);
317     cfm->req = param->req;
318     // Set the device appearance
319     cfm->info.appearance = APP_HID_MOUSE_APPEARANCE;
320
321     // Send message
322     ke_msg_send(cfm);
323   }
324   break;
```

3. 在新增 HID Profile 时，填入当前的应用类型，这位于 app_hid.c 中的 app_hid_add_hids 函式，使用者可以根据当前需求填入键盘或是鼠标的宏定义，但要记得一定要有 Protocol Mode 才能正常使用。

图 3. GAP Client Gets Device Apperance

```
287   // Set parameters
288   db_cfg = (struct hogpd_db_cfg *)req->param;
289
290   // Only one HIDS instance is useful
291   db_cfg->hids_nb = 1;
292
293   /**************************************************/
294   // The device type
295   db_cfg->cfg[0].svc_features = HOGPD_CFG_MOUSE | HOGPD_CFG_PROTO_MODE;
296
297   // Only one Report Characteristic is requested
298   db_cfg->cfg[0].report_nb = 8;
```

2.3 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

wb415_hogp_bt_demo\projects\ble_app_remote

wb415_hogp_mcu_demo\utilities\wb415_hogp_mcu_demo\mdk_v5

2.4 软件设计

1) 配置流程

a) MCU 端

- 配置 USER KEY 作为击键
- 编写 USART3 通讯函数
- 轮询 USER KEY 是否按下
- 根据需求去配置 USER KEY 按下后发出的 AT command

b) BT 端

- 将 HOGP 的 Profile 加入 database
- 编写应用层与 GATT 之间的界面
- 在应用层中轮询 AT command
- 透过 HOGP 将不同的 key function 发给 host 端

2) 代码介绍

- MCU 端
  - main 函数代码描述

```
int main(void)
{
  gpio_init_type gpio_init_struct;
#ifdef BT_FLASH_WR_TEST
  uint8_t data;
#endif

  /* 配置系统时钟 */
  system_clock_config();

  /* 初始化开发板资源 */
  at32_board_init();
  /* 初始化 USER Key */
  at32_button_init();

  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
  nvic_irq_enable(USART3_IRQn, 0, 0);
  nvic_irq_enable(USART2_IRQn, 0, 1);

  /* USART2 configured as follow:
        - BaudRate = 115200 baud
        - Word Length = 8 Bits
        - One Stop Bit
        - No parity
        - Hardware flow control disabled (RTS and CTS signals)
        - Receive and transmit enabled
  */
  crm_periph_clock_enable(CRM_USART3_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_IOMUX_PERIPH_CLOCK, TRUE);
```

```
gpio_pin_remap_config(USART3_GMUX_0010, TRUE);
gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init_struct.gpio_out_type  = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_mode = GPIO_MODE_MUX;
gpio_init_struct.gpio_pins = GPIO_PINS_7;
gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
gpio_init(GPIOA, &gpio_init_struct);

gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
gpio_init_struct.gpio_out_type  = GPIO_OUTPUT_PUSH_PULL;
gpio_init_struct.gpio_mode = GPIO_MODE_INPUT;
gpio_init_struct.gpio_pins = GPIO_PINS_6;
gpio_init_struct.gpio_pull = GPIO_PULL_UP;
gpio_init(GPIOA, &gpio_init_struct);

uart_print_init(115200);

usart_init(USART3, 115200, USART_DATA_8BITS, USART_STOP_1_BIT);
usart_transmitter_enable(USART3, TRUE);
usart_receiver_enable(USART3, TRUE);



/* Enable the USARTx Interrupt */
usart_interrupt_enable(USART3, USART_RDBF_INT, TRUE);
usart_enable(USART3, TRUE);

printf("System start up...\r\n");

while (1)
{
  /* 轮询 USER Key*/
  if(at32_button_press() == USER_BUTTON)
  {
    /* 发送 back key AT command */
    at_cmd_send(AT_CMD_WWW_BACK);
    /* 等候 BT 端的回应 */
    while(cmd_rsp_get(AT_RST_WWW_BACK_OK) != RSP_OK);
```

```
    }
  }
}
```

- BT 端

原则上不需要去修改 Profile 层的内容，由 APP 层去设定 HID 装置的内容即可

- 初始化 HOGP 装置变量

```
void app_hid_init(void)
{
    // Reset the environment
    memset(&app_hid_env, 0, sizeof(app_hid_env));
    app_hid_env.state = APP_HID_IDLE;
    app_hid_set_send_flag(true);
}
```

- 将 HOGP 加入 database

```
void app_hid_add_hids(void)
{
    struct hogpd_db_cfg *db_cfg;
    // Prepare the HOGPD_CREATE_DB_REQ message
    struct gapm_profile_task_add_cmd *req = KE_MSG_ALLOC_DYN(GAPM_PROFILE_TASK_ADD_CMD,
                TASK_GAPM, TASK_APP,
                gapm_profile_task_add_cmd, sizeof(struct hogpd_db_cfg));

    // Fill message
    req->operation = GAPM_PROFILE_TASK_ADD;
    req->sec_lvl = 0;
    req->prf_task_id = TASK_ID_HOGPD;
    req->app_task = TASK_APP;
    req->start_hdl = 0;

    // Set parameters
    db_cfg = (struct hogpd_db_cfg *)req->param;

    // Only one HIDS instance is useful
    db_cfg->hids_nb = 1;
```

```
/*****************************************************/
// The device type
db_cfg->cfg[0].svc_features = HOGPD_CFG_KEYBOARD | HOGPD_CFG_PROTO_MODE;

// Only one Report Characteristic is requested
db_cfg->cfg[0].report_nb = 8;

db_cfg->cfg[0].report_id[0] = HIDS_KB_REPORT_ID; // standard key

// The report is an input report
db_cfg->cfg[0].report_char_cfg[0] = HOGPD_CFG_REPORT_IN;

/*****************************************************/
db_cfg->cfg[0].report_id[1] = HIDS_KB_REPORT_ID;

// The report is an input report
db_cfg->cfg[0].report_char_cfg[1] = HOGPD_CFG_REPORT_OUT;

/*****************************************************/
db_cfg->cfg[0].report_id[2] = HIDS_MOUSE_REPORT_ID; // mouse

// The report is an input report
db_cfg->cfg[0].report_char_cfg[2] = HOGPD_CFG_REPORT_IN;

/*****************************************************/
db_cfg->cfg[0].report_id[3] = RMC_VENDOR_REPORT_ID_1; // voice 1

// The report is an input report
db_cfg->cfg[0].report_char_cfg[3] = HOGPD_CFG_REPORT_IN;
/*****************************************************/

db_cfg->cfg[0].report_id[4] = RMC_VENDOR_REPORT_ID_2; // voice 2

// The report is an input report
db_cfg->cfg[0].report_char_cfg[4] = HOGPD_CFG_REPORT_IN;

/*****************************************************/
db_cfg->cfg[0].report_id[5] = RMC_VENDOR_REPORT_ID_2;
```

```
        // The report is an input report
        db_cfg->cfg[0].report_char_cfg[5] = HOGPD_CFG_REPORT_OUT;
        /*****************************************************/

        db_cfg->cfg[0].report_id[6] = HIDS_MM_KB_REPORT_ID; // media    key

        // The report is an input report
        db_cfg->cfg[0].report_char_cfg[6] = HOGPD_CFG_REPORT_IN;
        /*****************************************************/

        db_cfg->cfg[0].report_id[7] = RMC_SENSORS_DATA_REPORT_ID; // sensor

        // The report is an input report
        db_cfg->cfg[0].report_char_cfg[7] = HOGPD_CFG_REPORT_IN;
        /*****************************************************/

        // HID Information
        db_cfg->cfg[0].hid_info.bcdHID = 0x0111; // HID Version 1.11
        db_cfg->cfg[0].hid_info.bCountryCode = 0x00;
        db_cfg->cfg[0].hid_info.flags = HIDS_REMOTE_WAKE_CAPABLE | HIDS_NORM_CONNECTABLE;

        // Send the message
        ke_msg_send(req);
}
```

在 main 函数之中的 while loop 不断轮询 app_user_entry()来确认有无收到 AT command，如果收到有收到 AT command 则执行该命令的内容。

- 解析 AT command 并执行对应的程序

```
void app_user_entry(void)
{
    uint8_t without_prefix_len;

    // GPIO_int_enable();

    if (ke_state_get(TASK_APP) == APPM_READY)
    {
        UART_PRINTF("start advertising\r\n");
```

```
        appm_start_advertising();
    }

    if (uart_rx_done == 1)
    {
        // uint8_t baud_change = 0;
        uint8_t len;
        uint8_t rsp_code;
        // uint8_t idx;
        extern uint8_t rxdata_buffer_len;
        at_prefix_t *prefix_cmd;
        uint8_t w_flash_buf[2];

        // len = strlen((char*)rxdata_buffer);
        len = rxdata_buffer_len;
        rxdata_buffer_len = 0;
        if (rxdata_buffer[len - 1] == '\n')
        {
            // AT command finish
            // UART_PRINTF("finish\r\n");
            memcpy(&AT_cmd_buf[recv_AT_cmd_idx], rxdata_buffer, len);
            // UART_PRINTF("%s\r\n",AT_cmd_buf);
            AT_cmd_len += len;
            recv_AT_cmd_idx = 0;
        }
        else
        {
            // command not finish
            memcpy(&AT_cmd_buf[recv_AT_cmd_idx], rxdata_buffer, len);
            recv_AT_cmd_idx = len;
            AT_cmd_len += len;
            uart_rx_done = 0;
            // UART_PRINTF("not finish\r\n");
            return;
        }

        // dispatch AT-COMMAND
        prefix_cmd = at_result_to_prefix((char *)rxdata_buffer, len);
```

```
            uart_rx_done = 0;
            without_prefix_len = AT_cmd_len - prefix_cmd->prefix_len;
            //                          UART_PRINTF("len : %d\r\n",without_prefix_len);//len are include "\r\n" char
            if (prefix_cmd == NULL)
            {
#ifdef used_WB415
                UART_SEND_DATA("@");
#endif
                UART_SEND_DATA("ERROR\r\n");
            }
            else if (prefix_cmd != NULL)
            {
                rsp_code = cmd_rsp[prefix_cmd->code];
                switch (prefix_cmd->code)
                {
                case AT_RESULT_AT:
                    // do nothing

#ifdef used_WB415
                    UART_SEND_DATA("@");
#endif
                    UART_SEND_DATA("%s\r\n", get_at_rsp(rsp_code));
                    break;
                case AT_RESULT_BAUDS1:
#ifdef used_WB415
                    UART_SEND_DATA("@");
#endif
                    UART_SEND_DATA("%s\r\n", get_at_rsp(rsp_code)); // 2018/12/25 fix4,use old baudrate send
response and change new baudrate
                    cpu_delay(15);
                    uart_init(9600);
                    break;
                case AT_RESULT_BAUDS2:
#ifdef used_WB415
                    UART_SEND_DATA("@");
#endif
                    UART_SEND_DATA("%s\r\n", get_at_rsp(rsp_code)); // 2018/12/25 fix4,use old baudrate send
response and change new baudrate
```

```
                    cpu_delay(15);
                    uart_init(19200);
                    UART_SEND_DATA("%s\r\n", get_at_rsp(rsp_code));
                    break;
                case AT_RESULT_BAUDS3:
#ifdef used_WB415
                    UART_SEND_DATA("@");
#endif
                    UART_SEND_DATA("%s\r\n", get_at_rsp(rsp_code)); // 2018/12/25 fix4,use old baudrate send response and change new baudrate
                    cpu_delay(15);
                    uart_init(38400);
                    break;
                case AT_RESULT_BAUDS4:
#ifdef used_WB415
                    UART_SEND_DATA("@");
#endif
                    UART_SEND_DATA("%s\r\n", get_at_rsp(rsp_code)); // 2018/12/25 fix4,use old baudrate send response and change new baudrate
                    cpu_delay(15);
                    uart_init(57600);
                    break;
                case AT_RESULT_BAUDS5:
#ifdef used_WB415
                    UART_SEND_DATA("@");
#endif
                    UART_SEND_DATA("%s\r\n", get_at_rsp(rsp_code)); // 2018/12/25 fix4,use old baudrate send response and change new baudrate
                    cpu_delay(15);
                    uart_init(115200);
                    break;
                case AT_RESULT_VOL0:
#ifdef used_WB415
                    UART_SEND_DATA("@");
#endif
                    app_hid_send_report(&key_copy[18], 2);
                    UART_SEND_DATA("%s\r\n", get_at_rsp(rsp_code));
                    cpu_delay(15);
                    break;
```

```
                case AT_RESULT_VOL1:
#ifdef used_WB415
                    UART_SEND_DATA("@");
#endif
                    app_hid_send_report(&key_copy[20], 2);
                    UART_SEND_DATA("%s\r\n", get_at_rsp(rsp_code));
                    cpu_delay(15);
                    break;
                case AT_RESULT_WWW_BACK:
#ifdef used_WB415
                    UART_SEND_DATA("@");
#endif
                    app_hid_send_report(&key_copy[0], 2);
                    UART_SEND_DATA("%s\r\n", get_at_rsp(rsp_code));
                    cpu_delay(15);
                    break;
                    case AT_RESULT_MOUSE:
#ifdef used_WB415
                    UART_SEND_DATA("@");
#endif
                    report.b = 0x00;
                    report.w = 0x00;
                    report.x = 0x7F;
                    report.y = 0x7F;
                    app_hid_send_mouse_report(report);
                    UART_SEND_DATA("%s\r\n", get_at_rsp(rsp_code));
                    cpu_delay(15);
                    break;
                }
            }
            AT_cmd_len = 0;
        }
}
```

2.5 实验效果

- 按下 AT-START 板上的 USER KEY 发送 www back 按键，则浏览器页面返回上一页
- 按下 AT-START 板上的 USER KEY 发送 volume up 按键，则音量提高（需要修改 AT command）

- 按下 AT-START 板上的 USER KEY 发送 volume down 按键，则音量降低（需要修改 AT command）

2.6 按键定义

本应用指南以多媒体按键为例，在代码中定义了多媒体按键值的数组，用户可以在 app_key.c 中找到以下代码：

- 多媒体按键值定义

```
const uint8_t media_key[47][2] =
    {
        {0x24, 0x02}, // WWW back    0
        {0x25, 0x02}, // WWW forward 1
        {0x26, 0x02}, // WWW Stop    2
        {0x27, 0x02}, // WWW Refresh 3
        {0x21, 0x02}, // WWW Search 4
        {0x2A, 0x02}, // WWW Favorites    5
        {0x23, 0x02}, // WWW Home 6
        {0x8A, 0x01}, // Mail    7
        {0xE2, 0x00}, // Mute 8
        {0xEA, 0x00}, // Volume-    9
        {0xE9, 0x00}, // Volume+     10
        {0xCD, 0x00}, // Play/Pause    11
        {0xB7, 0x00}, // Stop    12
        {0xB6, 0x00}, // Prev Track 13
        {0xB5, 0x00}, // Next Track    14
        {0x83, 0x01}, // Media Select    15
        {0x94, 0x01}, // My Computer     16
        {0x92, 0x01}, // Calculator     17
        {0x09, 0x02}, // More Info    18
        {0xB2, 0x00}, // Record         19
        {0xB3, 0x00}, // Forward        20
        {0xB4, 0x00}, // Rewind         21
        {0x8D, 0x00}, // Guide            22
        {0x04, 0x00}, //<Reserved>     23
        {0x30, 0x00}, // Eject(Mac)     24
        {0x07, 0x03}, // H7       25
        {0x0A, 0x03}, // H10      26lightness+
        {0x0B, 0x03}, // H11      27lightness-
```

```
        {0xb1, 0x01}, // photo 28
        {0xb8, 0x00}, // touchkey 29
        {0x14, 0x03}, // H20     30
        {0x01, 0x03}, // H1     31
        {0x02, 0x03}, // H2     32
        {0x03, 0x03}, // H3      33
        {0x04, 0x03}, // H4      34
        {0x05, 0x03}, // H5      35
        {0x06, 0x03}, // H6      36
        {0x08, 0x03}, // H8      37
        {0x09, 0x03}, // H9      38
        {0x0C, 0x03}, // H12     39
        {0x0D, 0x03}, // H13     40
        {0x0E, 0x03}, // H14     41
        {0x0F, 0x03}, // H15     42
        {0x10, 0x03}, // H16     43
        {0x11, 0x03}, // H17     44
        {0x12, 0x03}, // H18     45
        {0x13, 0x03}, // H19     46
};
```

在 app.c 中的 app_user_entry 函数，每当添加一个新的按键实例，必须透过 app_hid_send_report 这个函数，将按键值及长度发送给主机端，主机才能根据按键值做出对应的行为；举例来说，要发出降低音量的按键功能，从上面的注释可以看到 volume down 对应到第九列，是整个数组的第 18 个数，因此调用发送函数时写成 app_hid_send_report(&key_copy[18], 2)，其他按键也是如此使用。

2.7 HOGP 相关 AT command

本应用指南中，已经建立部分的 AT command，如下表：

表 2. HOGP 相关 AT command

AT command	Definition	Return Code
AT+VOL0	降低音量	OKVOL0/ERROR
AT+VOL1	提升音量	OKVOL1/ERROR
AT+WWWBACK	网页返回上一页	OKWWWBACK/ERROR
AT+X127Y127W0B0	移动鼠标	OKMOUSE/ERROR

用户自行添加 AT command 时，需要在 MCU 端及 BT 端同时增加对应的实例；MCU 端要在 at_cmd.h 中新增 AT command 的请求及期待的响应；BT 端要在 app.h 中新增要解析的 AT command 字符串、command 的枚举以及回复的枚举，数量上如果对不上的话则发出的按键功能可能会不正常。

另外鼠目标 AT command 及位移都是填入 Dummy Data，使用者如果要反应真实的位移及实作按键、

滚轮等功能，需要自己由传感器读取数据，并修改鼠标的 AT command 让它可以携带真实参数，接收 AT command 的部份也要实做解析函数去取出 AT command 中的参数内容，并填入 Notification 之中。

3 文档版本历史

表 3. 文档版本历史

日期	版本	变更
2022.8.8	2.0.0	最初版本
2022.10.28	2.0.1	新增鼠标应用修改方式

重要通知 - 请仔细阅读

买方自行负责对本文所述雅特力产品和服务的选择和使用，雅特力概不承担与选择或使用本文所述雅特力产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为雅特力授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在雅特力的销售条款中另有说明，否则，雅特力对雅特力产品的使用和/或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途(及其依据任何司法管辖区的法律的对应情况)，或侵犯任何专利、版权或其他知识产权的默示保证。

雅特力产品并非设计或专门用于下列用途的产品：(A) 对安全性有特别要求的应用，如：生命支持、主动植入设备或对产品功能安全有要求的系统；(B) 航空应用；(C) 汽车应用或汽车环境；(D) 航天应用或航天环境，且/或(E) 武器。因雅特力产品不是为前述应用设计的，而采购商擅自将其用于前述应用，即使采购商向雅特力发出了书面通知，风险由购买者单独承担，并且独力负责在此类相关使用中满足所有法律和法规要求。

经销的雅特力产品如有不同于本文档中提出的声明和/或技术特点的规定，将立即导致雅特力针对本文所述雅特力产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大雅特力的任何责任。