Bluetooth®
Low Energy
Software Developer′s Guide

v1.0

Beken Corporation
博通集成电路(上海)有限公司
中国上海张江高科技园区
张东路 1387 科技领袖之都 41 栋
电话： (86)21 5108 6811
传真： (86)21 6087 1277

更改记录

版本号	日期	作者	注释
V1.0	2020-02-07	lnc	文档建立

1. 软件快速入门

1.1 、用户配置

在软件开发包每个工程下面都有一个 user_config.h 文件，这个文件主要是用来配置蓝牙的一些基本参数，如蓝牙名称、连接间隔、广播包数据、扫描响应包数据、驱动等，部分截图如下：

```
#define VIRTUAL_UART_H4TL          1
#define UART_PRINTF_ENABLE         1///串口打印开关
#define DEBUG_HW                   0///硬件调试
#define DEBUG_HW_DIGITAL           0///数字硬件调试
#define GPIO_DBG_MSG               0///DEBUG信息可以通过DEBUG_MSG函数输出到GPIO
#define DEBUG_RF_REG               0///RF调试，为1可以通过串口读写RF寄存器
#define LDO_MODE                   0///LDO工作模式

//DRIVER CONFIG
#define UART0_DRIVER                1
#define UART2_DRIVER                1
#define GPIO_DRIVER                 0
#define ADC_DRIVER                  1
#define I2C_DRIVER                  0
#define PWM_DRIVER                  0
#define USB_DRIVER                  0
#define SPI_DRIVER                  0
#define AON_RTC_DRIVER              0

#define ADC_CALIB                   0//////校准ADC需要给芯片一个稳定的供电压，然后算ADC参考电压

#define uart_printf              uart2_printf

/// Default Device Name
#define APP_DFLT_DEVICE_NAME            ("BK3633_BLE789")
#define APP_DFLT_DEVICE_NAME_LEN        (sizeof(APP_DFLT_DEVICE_NAME))

#define APP_SCNRSP_DATA         "\x09\xFF\x00\x60\x42\x4B\x2D\x42\x4C\x45"
#define APP_SCNRSP_DATA_LEN     (10)


/// Advertising channel map - 37, 38, 39
#define APP_ADV_CHMAP           (0x07)
/// Advertising minimum interval - 40ms (64*0.625ms)
#define APP_ADV_INT_MIN         (160 )
/// Advertising maximum interval - 40ms (64*0.625ms)
#define APP_ADV_INT_MAX         (160)
/// Fast advertising interval
#define APP_ADV_FAST_INT        (32)

//最小连接间隔
#define BLE_UAPDATA_MIN_INTVALUE        20
//最大连接间隔
#define BLE_UAPDATA_MAX_INTVALUE        40
//连接Latency
#define BLE_UAPDATA_LATENCY             0
//连接超时
#define BLE_UAPDATA_TIMEOUT             600
```

1.2、 常用的 API 及回调函数

1.2.1、启动一个软件定时器

```
/*********************************************************************
 *
 * timer_id ： 定时器的 ID，用来区分是哪个定时器
 * task_id ：  任务 ID
 * Delay ： 定时器时间，单位 10ms
```

```
 *
 ****************************************************************/

void ke_timer_set(ke_msg_id_t const timer_id, ke_task_id_t const task_id,
                                uint32_t delay)
```

例如：ke_timer_set(APP_PERIOD_TIMER,TASK_APP,10);

对应的 100ms 后执行的握手函数是：

```
static int app_period_timer_handler(ke_msg_id_t const msgid,
                                              void *param,
                                              ke_task_id_t const dest_id,
                                              ke_task_id_t const src_id)
```

1.2.2、清除一个软件定时器

```
/****************************************************************
 *
 * timer_id : 定时器 ID
 * task_id :  任务 ID
 *
 ****************************************************************/
void ke_timer_clear(ke_msg_id_t const timer_id, ke_task_id_t const
                                task_id)
```

1.2.3、获取一个软件定时器的状况

```
/****************************************************************
 *
 * timer_id : 定时器 ID
 * task_id :  任务 ID
 *
 ****************************************************************/
bool ke_timer_active(ke_msg_id_t const timer_id, ke_task_id_t const
task_id)
```

1.2.3、创建广播

```
static void appm_create_advertising(void)
```

1.2.4、删除广播

```
static void appm_delete_advertising (void)
```

1.2.5、设备主动发起广播

```
void appm_start_advertising(void)
```

1.2.6、设备主动停止广播

```
void appm_stop_advertising(void)
```

1.2.7、广播各个状态处理

```
void appm_adv_fsm_next(void)
```

1.2.8、设备主动断开连接

```
void appm_disconnect(void)
```

1.2.9、设备连接成功

```
static int gapc_connection_req_ind_handler(ke_msg_id_t const msgid,
                        struct gapc_connection_req_ind const *param,
                        ke_task_id_t const dest_id,
                        ke_task_id_t const src_id)
```

1.2.10、设备断开

```
static int gapc_disconnect_ind_handler(ke_msg_id_t const msgid,
                        struct gapc_disconnect_ind const *param,
                        ke_task_id_t const dest_id,
                        ke_task_id_t const src_id)
```

1.2.11、数据发送，以 FCC2 为例

```
void app_fcc2_send_ntf(uint8_t conidx,uint16_t len,uint8_t* buf)
```

1.2.12、数据接收，以 FCC2 为例

```
static int fcc1_writer_cmd_handler(ke_msg_id_t const msgid,
                        struct fcc0s_fcc1_writer_ind *param,
                        ke_task_id_t const dest_id,
                        ke_task_id_t const src_id)
```

1.2.13、数据发送完成事件

GATTC_CMP_EVT 事件对 GATT 操作状态进行处理，当发送完数据之后，协议栈会上报 GATTC_CMP_EVT 事件（在每个 profile 对应的 xxx_task.c 中），可以在这个事件中判断数据发送的具体状态。

```
static int gattc_cmp_evt_handler(ke_msg_id_t const msgid,
                            struct gattc_cmp_evt const *param,
```

```
                                ke_task_id_t const dest_id,
                                ke_task_id_t const src_id)

//Command complete event data structure
struct gattc_cmp_evt
{
    /// GATT request type
    uint8_t operation;
    /// Status of the request
    uint8_t status;
    /// operation sequence number - provided when operation is started
    uint16_t seq_num;
};
```

GAP_ERR_NO_ERROR：成功
其他：失败

1.2.14、更新连接间隔请求

```
static int app_update_conn_param_req_ind_handler (ke_msg_id_t const msgid,
                                                  const struct gapc_param_update_req_ind  *param,
                                                  ke_task_id_t const dest_id,
                                                  ke_task_id_t const src_id)
```

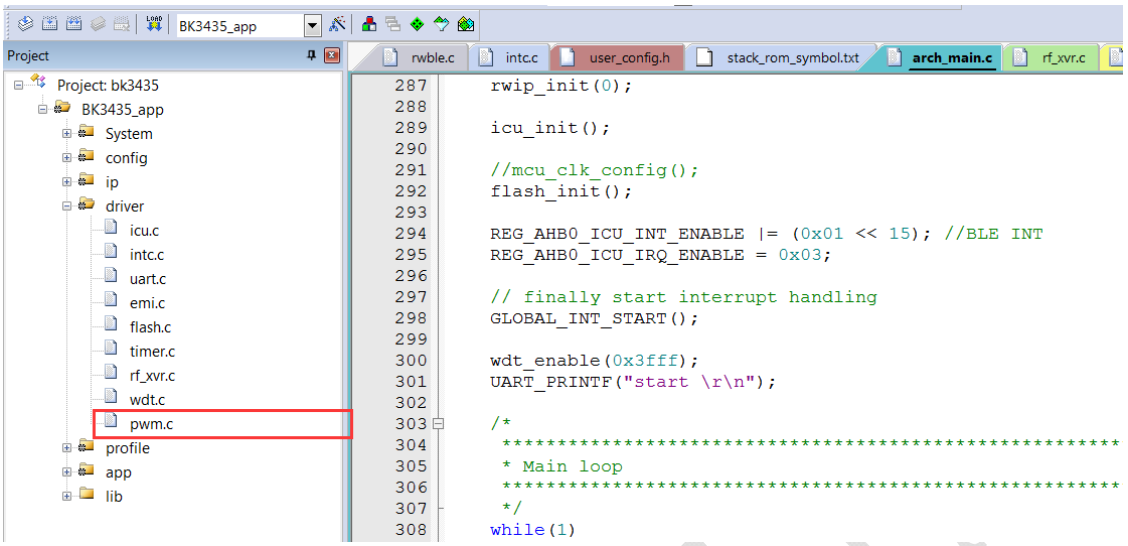
1.3 、外设驱动使用举例

以下通过在 ble_app_gatt 这个工程中添加 PWM 驱动来说明如何添加一个驱动到对应的工程项目中：

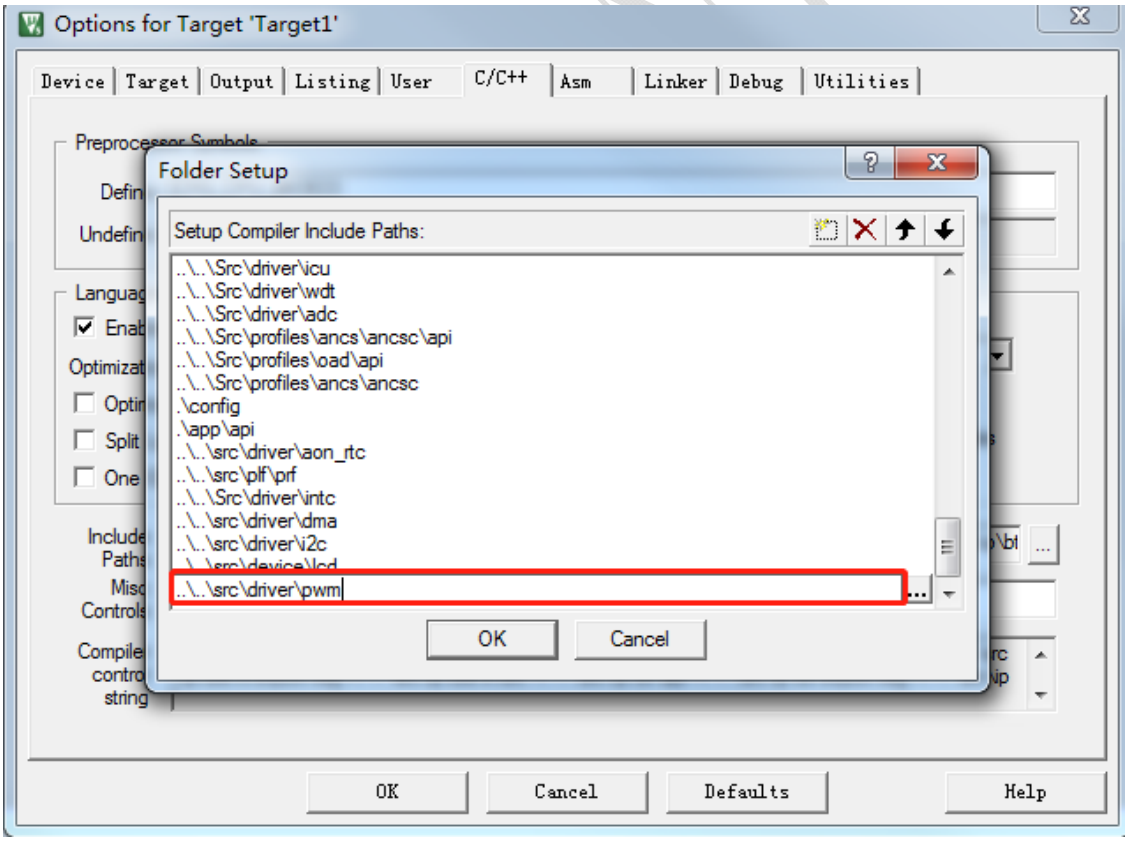
① 在工程对应的\projects\ble_app_gatt\config 目录下，打开 user_config.h 文件，打开对应的宏定义，如下：

```
//DRIVER CONFIG
#define UART0_DRIVER                1
#define UART2_DRIVER                1
#define GPIO_DRIVER                 0
#define ADC_DRIVER                  1
#define I2C_DRIVER                  0
#define PWM_DRIVER                  1
#define USB_DRIVER                  0
#define SPI_DRIVER                  0
#define AON_RTC_DRIVER              0
```

② 使用 keil 5.12 打开 ble_app_gatt 工程，在 keil 中的 driver 下添加 pwm.c 源文件如下：



③ 在 keil 工程配置中添加相关路径，如下：

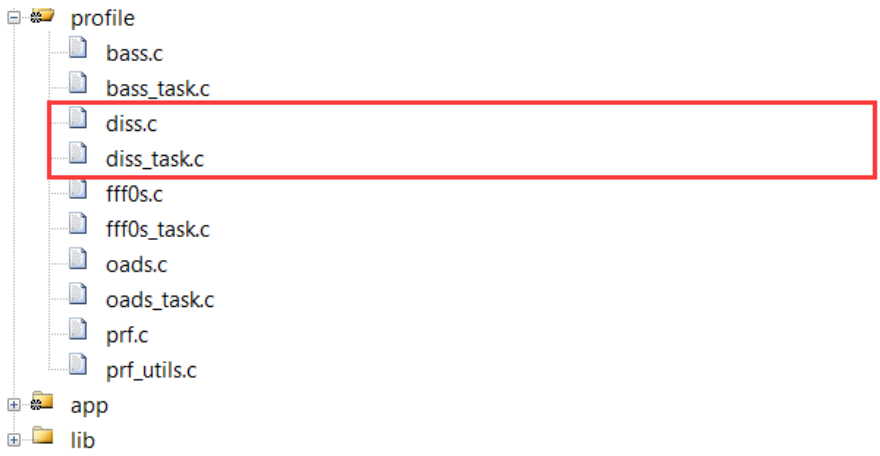


在 main 函数内 GLOBAL_INT_START 之前调用测试函数
void pwm_pwmmode_test(void)

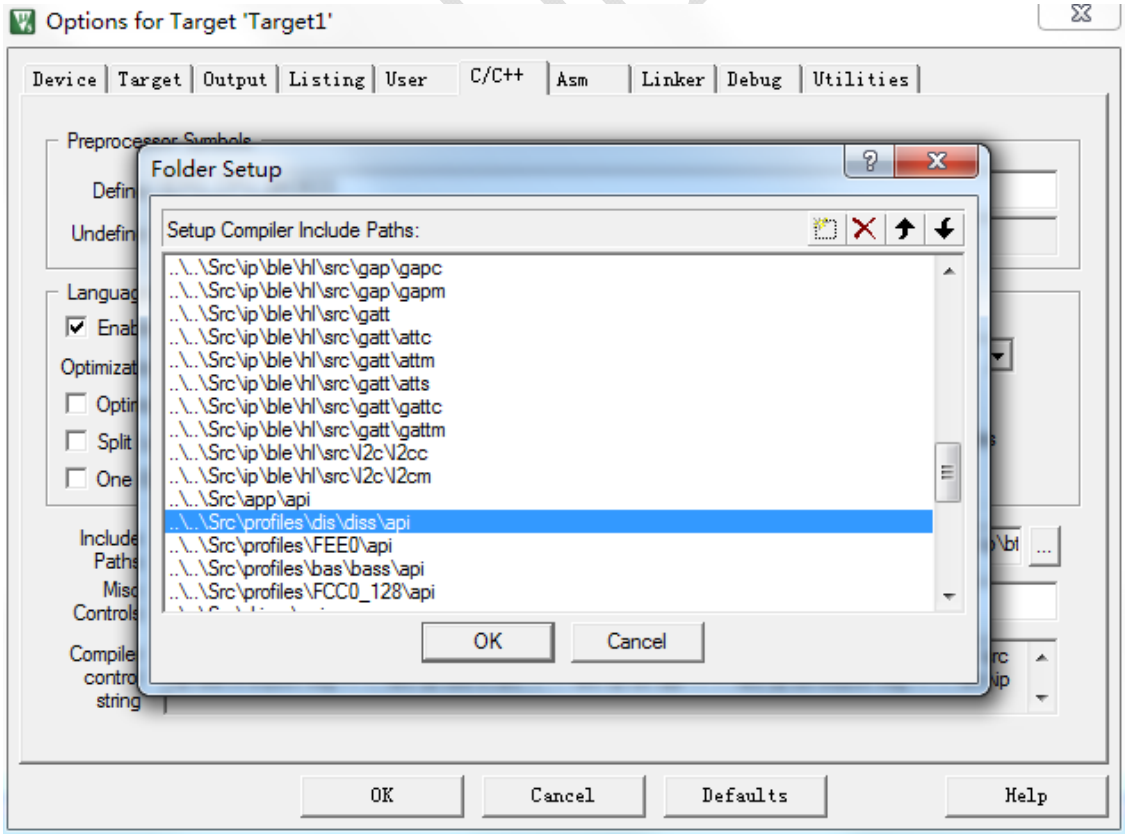
1.4 添加一个服务

本部分将通过在 ble_app_gatt 工程中添加一个 device information 服务来说明具体的添加步骤。

① 使用 keil 5.12 打开对应工程，在工程目录 profile 目录下添加指定 profile 相关源文件，本例中是 diss.c 和 diss_task.c，如下：



② 在 keil C/C++选项卡下面添加对应 profile 的路径，如下：



③ 在 app_task.c 文件中找到 appm_msg_handler 函数，在其中添加设备信息服务获取 handler 的函数，如下：

```
case (TASK_ID_BASS):
{
    // Call the Battery Module
    msg_pol = appm_get_handler(&app_batt_table_handler, msgid, param, src_id);
} break;
```

④ 在 app.c 文件中的服务列表中添加设备信息服务，如下：

```
/// List of service to add in the database
enum appm_svc_list
{
    APPM_SVC_FFF0,
    APPM_SVC_DIS,
    APPM_SVC_BATT,
    APPM_SVC_OADS,
    APPM_SVC_LIST_STOP ,
};
```

⑤ 在 app.c 文件中的函数列表中增加像 db 添加设备信息服务的函数，如下：

```
/// List of functions used to create the database
static const appm_add_svc_func_t appm_add_svc_func_list[APPM_SVC_LIST_STOP] =
{
    (appm_add_svc_func_t)app_fff0_add_fff0s,
    (appm_add_svc_func_t)app_dis_add_dis,
    (appm_add_svc_func_t)app_batt_add_bas,
    (appm_add_svc_func_t)app_oad_add_oads,
};
```

⑥ 在 app.c 文件中找到 appm_init 函数，在其中添加 app_dis_init()函数，如下：

```
// Device Information Module
app_dis_init();

// Battery Module
app_batt_init();

app_oads_init();
```

⑦ 打开 rwprf_config.h 文件，将其中的 CFG_PRF_DISS 宏定义设置为 1，如下：

```
//   <e> CFG_PRF_DISS
//   <i> Battery Service Server Role
//   </e>
#if ( 1 )
#define CFG_PRF_DISS
#endif
```

⑧ 重新编译工程，通过手机搜索连接，确认设备信息服务是否已经添加成功。

1.5 添加一个自定义消息

这部分通过在工程中添加一个连接参数更新的消息来描述在系统中如何添加一个自定义消息。

① 在 app_task.h 中定义自定义消息 ID，如：APP_PARAM_UPDATE_REQ_IND。

```
enum appm_msg
{
    APPM_DUMMY_MSG = TASK_FIRST_MSG(TASK_ID_APP),
    APP_PARAM_UPDATE_REQ_IND,
};
```

② 在 const struct ke_msg_handler appm_default_state[]中添加自定义消息 ID 及对应的 handler，如下，

```
{APP_PARAM_UPDATE_REQ_IND,      (ke_msg_func_t)gapc_update_conn_param_req_ind_handler},
```

③ 实现 gapc_update_conn_param_req_ind_handler 函数

```
static int gapc_update_conn_param_req_ind_handler (ke_msg_id_t const msgid,
                                        const struct gapc_param_update_req_ind  *param,
                                        ke_task_id_t const dest_id,
                                        ke_task_id_t const src_id)
{

    UART_PRINTF("slave send param_update_req\r\n");
    struct gapc_conn_param  up_param;

    up_param.intv_min    = BLE_UAPDATA_MIN_INTVALUE;//最小连接间隔
    up_param.intv_max    = BLE_UAPDATA_MAX_INTVALUE;//最大连接间隔
    up_param.latency     = BLE_UAPDATA_LATENCY;  //latency
    up_param.time_out    = BLE_UAPDATA_TIMEOUT; //连接超时

    appm_update_param(&up_param);

    return KE_MSG_CONSUMED;
}
```

④ 在设备建立连接之后，设置一个定时器进行连接参数更新

```
ke_timer_set(APP_PARAM_UPDATE_REQ_IND,TASK_APP,100);
```

⑤ 连接参数更新状态指示

```
static int gapc_cmp_evt_handler(ke_msg_id_t const msgid,
                                struct gapc_cmp_evt const *param,
                                ke_task_id_t const dest_id,
                                ke_task_id_t const src_id)
{
    UART_PRINTF("gapc_cmp_evt_handler operation = %x\r\n",param->operation);
    switch(param->operation)
    {
        case (GAPC_UPDATE_PARAMS):  //0x09
        {
            if (param->status != GAP_ERR_NO_ERROR)
            {
                UART_PRINTF("gapc update params fail !\r\n");
            }
            else
            {
                UART_PRINTF("gapc update params ok !\r\n");
            }

        } break;
```

⑥ 成功更新连接参数后，协议栈会上报 GAPC_PARAM_UPDATED_IND 消息，并将最终更新的连接参数上报到应用层。

```
{GAPC_PARAM_UPDATED_IND,        (ke_msg_func_t)gapc_param_updated_ind_handler},

static int gapc_param_updated_ind_handler (ke_msg_id_t const msgid,
                                           const struct gapc_param_updated_ind  *param,
                                           ke_task_id_t const dest_id,
                                           ke_task_id_t const src_id)
{
    UART_PRINTF("%s \r\n", __func__);
    UART_PRINTF("con_interval = %d\r\n",param->con_interval);
    UART_PRINTF("con_latency = %d\r\n",param->con_latency);
    UART_PRINTF("sup_to = %d\r\n",param->sup_to);

    return KE_MSG_CONSUMED;
}
```

⑦ 如果是从机端不更新连接参数，主机端更新连接参数，我们可以设定从机端是否接受主机发过来的连接参数。当主机发起连接参数更新请求或，协议栈会像从设备应用层发送 GAPC_PARAM_UPDATE_REQ_IND 指示，如下，

```
{GAPC_PARAM_UPDATE_REQ_IND,     (ke_msg_func_t)gapc_param_update_req_ind_handler},

static int gapc_param_update_req_ind_handler(ke_msg_id_t const msgid,
                                struct gapc_param_update_req_ind const *param,
                                ke_task_id_t const dest_id,
                                ke_task_id_t const src_id)
{
    UART_PRINTF("%s \r\n", __func__);
    // Prepare the GAPC_PARAM_UPDATE_CFM message
    struct gapc_param_update_cfm *cfm = KE_MSG_ALLOC(GAPC_PARAM_UPDATE_CFM,
                                        src_id, dest_id,
                                        gapc_param_update_cfm);

    cfm->ce_len_max = 0xffff;
    cfm->ce_len_min = 0xffff;
    cfm->accept = true;

    // Send message
    ke_msg_send(cfm);

    return (KE_MSG_CONSUMED);
} ? end gapc_param_update_req_ind_handler ?
```

其中 cfm->accept = true 表示从设备接受主设备发起的连接参数更新。

1.6 配对方式修改

关于配对的相关操作都在 app_sec.c 这个文件中。

static int gapc_bond_req_ind_handler(ke_msg_id_t const msgid,
struct gapc_bond_req_ind const *param,
ke_task_id_t const dest_id,
ke_task_id_t const src_id)

在这个函数中对应如下几个参数

```
// Pairing Features
cfm->data.pairing_feat.auth      = GAP_AUTH_REQ_NO_MITM_BOND;
cfm->data.pairing_feat.iocap     = GAP_IO_CAP_NO_INPUT_NO_OUTPUT;
cfm->data.pairing_feat.key_size  = 16;
cfm->data.pairing_feat.oob       = GAP_OOB_AUTH_DATA_NOT_PRESENT;
cfm->data.pairing_feat.sec_req   = GAP_NO_SEC;
cfm->data.pairing_feat.ikey_dist = GAP_KDIST_ENCKEY| GAP_KDIST_IDKEY;
cfm->data.pairing_feat.rkey_dist = GAP_KDIST_ENCKEY| GAP_KDIST_IDKEY;
```

修改了cfm->data.pairing_feat.auth这个参数之后对应在
void app_sec_send_security_req(uint8_t conidx)这个函数中的cmd->auth 这个参数也要保持一致。

例如在 ble_app_ancs 这个程序默认是手机端确认的方式配对，如果需要改成 pincode 配对，那么需要修改如下几个参数：

```
cmd->auth                      = GAP_AUTH_REQ_MITM_BOND;
cfm->data.pairing_feat.auth    = GAP_AUTH_REQ_MITM_BOND;
cfm->data.pairing_feat.iocap   = GAP_IO_CAP_DISPLAY_ONLY;
```

对应的配对 pincode 如下为 123456 就为配对时需要输入的密码，可自行修改：

```
case (GAPC_TK_EXCH):
{
    // Generate a PIN Code- (Between 100000 and 999999)
    uint32_t pin_code =123456;// (100000 + (co_rand_word()%900000));
```