RT58x Multi-Protocol Application Guide V1.4

About this Document

This document supports “RT58x_SDK_v0.3.1” and later version.

Table of Contents

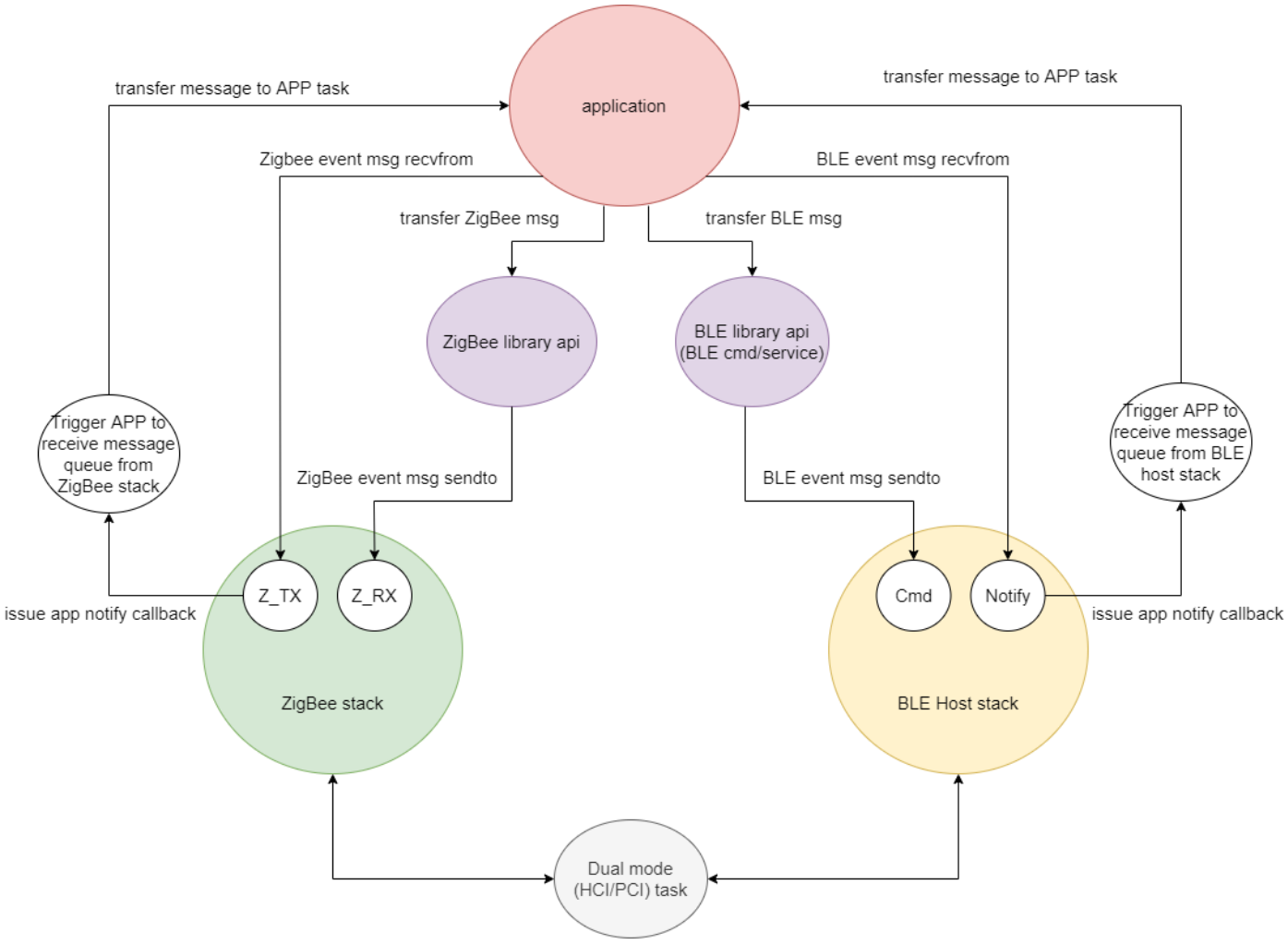
1. Introduction

This document describes the Zigbee/BLE multi-protocol architecture and demonstration (ZigBee switch and BLE Peripheral TRSP).

2. System Architecture

This section presents an overview of the ZigBee/BLE multi-protocol software architecture.

2.1 Task architecture



2.1.1 Zigbee Stack Task

Rafael ‘s Zigbee stack sub-system.

2.1.2 BLE Host Task

Rafael’s BLE stack sub-system.

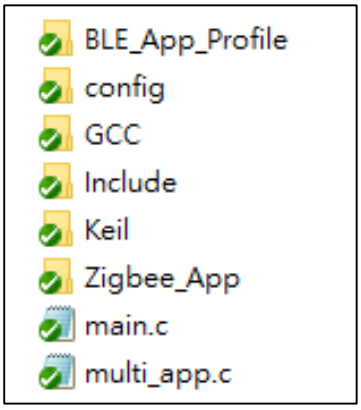
2.1.3 Dual mode task

Handle the BLE HCI command, event and ACL data for BLE host stack, also support IEEE 802.15.4 related packet transmit/receive for ZigBee stack

2.1.4 APP Tasks

Handle BSP, Application, ZCL, BLE event and product behavior.

2.2 File architecture



2.2.1 config

Configuration of project and FreeRTOS.

2.2.2 Include

Header files of application.

2.2.3 Keil

Keil project files.

2.2.4 GCC

GCC project files.

2.2.5 Zigbee_App

- *zigbee_app.c* :

  device context of application (Attribute, Cluster, Endpoint, Simple description)

- *zigbee_evt_handler.c:*

  Process event message from ZigBee stack library

- *zigbee_zcl_msg_handler.c:*

  Process ZCL event message from ZigBee stack library

2.2.6 BLE_App_Profile

- *ble_app.c* :

  device context of application (BLE service, event)

- *ble_profile_app.c:*

- BLE profile definition for application

- *ble_profile_def.c:*

  BLE profile definition (service combination of each link)

2.2.7 main.c

Main function of application.

2.2.8 multi_app.c

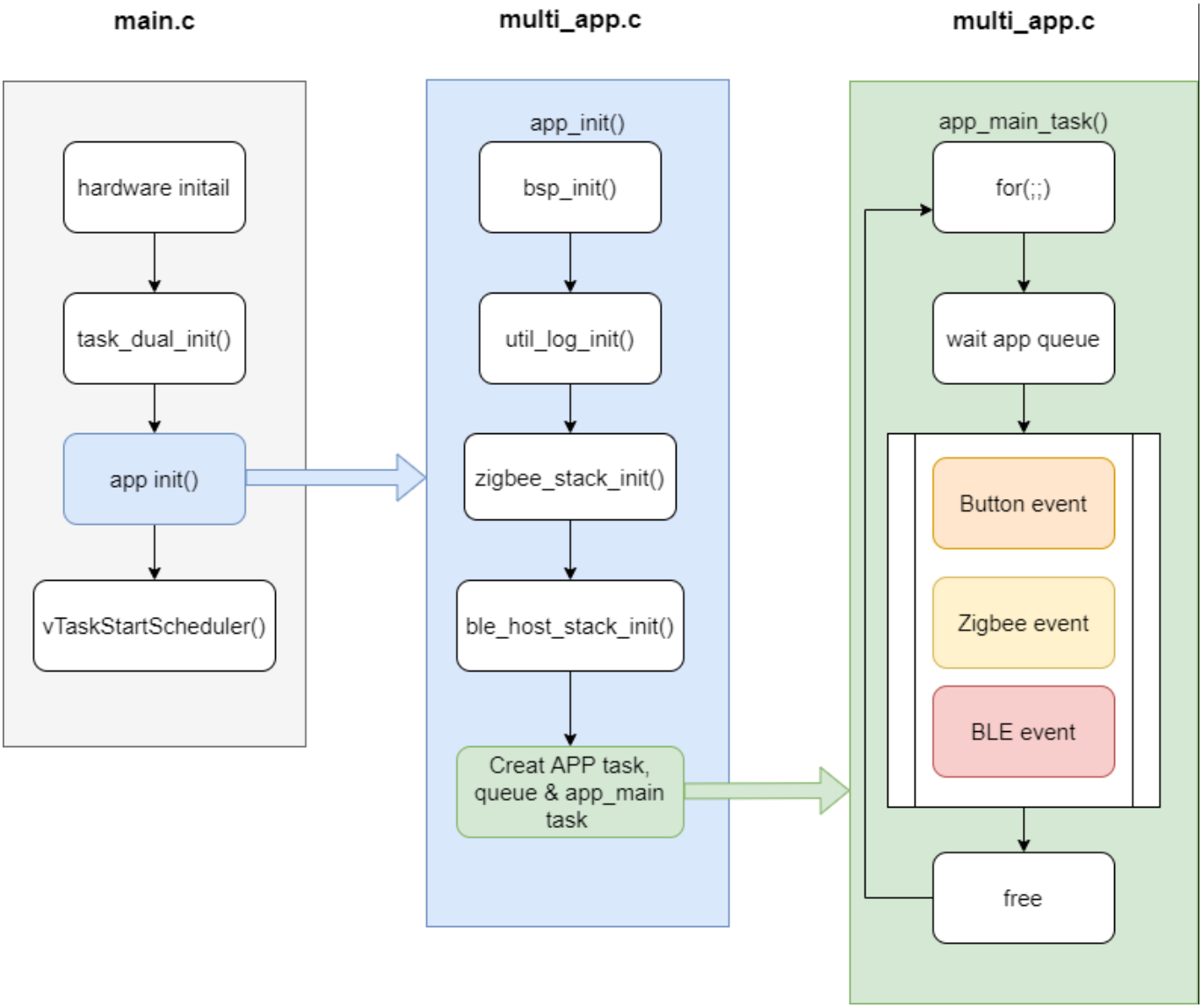
Task function of application that support BLE, ZigBee and BSP event

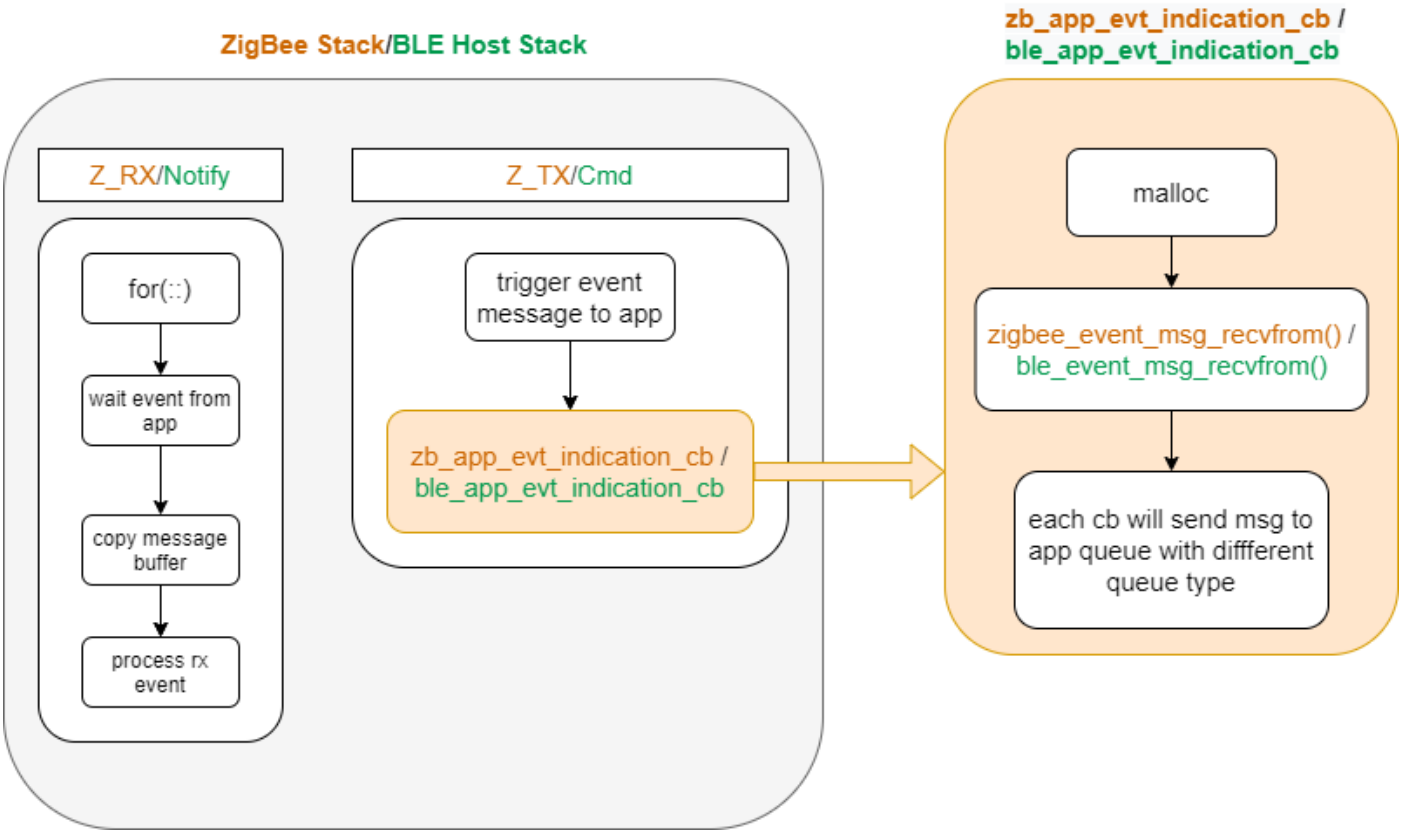
3. Application development

This section introduces the multi-protocol application flow. User could follow the instruction in this section to develop a customized multi-protocol application.

3.1 Application Flow

The following figure shows the multi-protocol application flow.





Application task may receive the message from “app queue” with 3 kinds of queue:

- QUEUE_TYPE_ZIGBEE
  Issue it to “zigbee_zcl_msg_handler()” or “zigbee_evt_handler()” function.
- QUEUE_TYPE_BLE
  Issue it to each BLE service handler or “ble_evt_handler()” function.
- QUEUE_TYPE_APP_REQ
  According to event type to issue it to ble_app_main() or zb_app_main() function.

3.2 Initialization

RT58x ZigBee application initialization process is shown as below:

1. Initial 802.15.4 & BLE HCI multi-protocol driver (task_dual_init())
2. Initial BSP(Optional).
3. Initial utility logging (Optional)
4. Initial CLI Console(Optional)
5. Register ZigBee/BLE device context and indication event callback.
6. Initial ZigBee/BLE stack.
7. Create application task and application message queue.

```
int main(void)
{
    /*we should set pinmux here or in SystemInit */
    init_default_pin_mux();

    Lpm_Set_Low_Power_Level(LOW_POWER_LEVEL_SLEEP0);
    Lpm_Enable_Low_Power_Wakeup(LOW_POWER_WAKEUP_GPIO);
    Lpm_Enable_Low_Power_Wakeup(LOW_POWER_WAKEUP_32K_TIMER);

    task_dual_init();
    app_init();
    /* Start the scheduler. */
    vTaskStartScheduler();
    while (1) {
    }

    return 0;
}
```

```
void app_init(void)
{
    /* Initil LED, Button, Console or UART */
    bsp_init((BSP_INIT_LEDS |
            BSP_INIT_BUTTONS|
            BSP_INIT_DEBUG_CONSOLE |
            BSP_INIT_UART), app_bsp_event_handle);

    /* Retarget stdout for utility & initial utility logging */
    utility_register_stdout(bsp_console_stdout_char, bsp_console_stdout_string);
    util_log_init();

    util_log_on(UTIL_LOG_PROTOCOL);

    gt_zb_app_cfg.p_zigbee_device_contex_t = &simple_desc_switch_ctx;
    gt_zb_app_cfg.pf_evt_indication = zb_app_evt_indication_cb;

    info_color(LOG_BLUE, "Initial ZigBee/BLE stack\n");

    zigbee_stack_init(&gt_zb_app_cfg);

    // BLE Stack init
    gt_ble_app_cfg.pf_evt_indication = ble_app_evt_indication_cb;
    ble_host_stack_init(&gt_ble_app_cfg);

    app_msg_q = xQueueCreate(16, sizeof(multi_app_queue_t));

    info("Create app task\n");
    xTaskCreate((TaskFunction_t)app_main_task, "app", 128, NULL, TASK_PRIORITY_APP, NULL);
}
```

3.3 Main loop

Implement ZigBee main loop “app_main_task ()” in multi_app.c file. Main loop runs main application and waits for event message from ZigBee/BLE stack and BSP.

Process application event status in “zb_app_main()/ble_app_main()” in zigbee_app.c/ble_app.c files. These functions control the application flow (e.g. ZigBee try to

join network, or BLE start ADV).

```
static void app_main_task(void *arg)
{
    multi_app_queue_t t_app_q;
    ble_err_t status;

    // BLE application init
    status = ble_init();
    if (status != BLE_ERR_OK)
    {
        info_color(LOG_RED, "[DEBUG_ERR] ble_app_ble_init() fail: %d\n", status);
        while (1);
    }

    //zigbee application init
    zigbee_app_evt_change(APP_INIT_EVT, FALSE);
    // start adv
    app_request_set(APP_TRSP_P_HOST_ID, APP_REQUEST_ADV_START, false);

    for(;;)
    {
        if(xQueueReceive(app_msg_q, &t_app_q, portMAX_DELAY)== pdTRUE)
        {
            switch(t_app_q.param_type)
            {
            case QUEUE_TYPE_APP_REQ:
            {
                switch (t_app_q.event)
                {
                case APP_QUEUE_ISR_BUTTON_EVT:
                    printf("button %d press\n", t_app_q.pin);
                    app_pin_parse(t_app_q.pin);
                break;

                case APP_QUEUE_BLE_EVT:
                {
                    ble_app_main(&t_app_q.param.ble_app_req);
                }
                break;

                case APP_QUEUE_ZIGBEE_EVT:
                {
                    zb_app_main(t_app_q.param.zb_app_req);
                }
                break;
                }

            }
            break;

            case QUEUE_TYPE_ZIGBEE:
            {
                zb_event_parse(t_app_q.param.pt_zb_tlv);
            }
            break;

            case QUEUE_TYPE_BLE:
            {
                ble_event_parse(t_app_q.param.pt_ble_tlv);

            }
            break;

            default:
                break;
            }
        }
    }
}
```

3.4 app_pin_parse

This function defined in multi_app.c file which is use to parse which GPIO pin been triggered

```
static void app_pin_parse(uint8_t pin)
{
    if(pin == BSP_EVENT_BUTTONS_0)
    {
        send_toggle();
    }
    else if (pin == BSP_EVENT_BUTTONS_1)
    {
        send_level_step(0);
    }
    else if(pin == BSP_EVENT_BUTTONS_2)
    {
        send_level_step(1);
    }
    else if(pin == BSP_EVENT_BUTTONS_3)
    {
        send_move_color();
    }
}
```

3.5 zb_app_main & ble_app_main

zb_app_main()is defined to process the request from function “zigbee_app_evt_change()”. The request types are like: start ZigBee network, try join ZigBee network or some specific application.

```
void zb_app_main(uint32_t event)
{
    switch (event)
    {
    case APP_INIT_EVT:

        if(zigbee_ed_nwk_start_request(ZIGBEE_CHANNEL_ALL_MASK(), false, 3000, !bsp_button_state_get(BSP_BUT-
TON_0)) != 0) /*0: SYS_ERR_OK*/
        {
            zigbee_app_evt_change(APP_INIT_EVT, FALSE);
        }
        break;

    case APP_NOT_JOINED_EVT:
        zigbee_join_request();
        break;

    case APP_START_REJOIN_EVT:
        zigbee_rejoin_request();
        break;

    case APP_LED_TOGGLE_EVT:
        info_color(LOG_YELLOW, "Toggle from BLE\n");
        send_toggle();
        break;

    case APP_LED_LEVEL_UP_EVT:
        info_color(LOG_YELLOW, "Level up from BLE\n");
        send_level_step(0);
        break;

    case APP_LED_LEVEL_DOWN_EVT:
        info_color(LOG_YELLOW, "Level down from BLE\n");
        send_level_step(1);
        break;

    case APP_LED_MOVE_TEMP_EVT:
        info_color(LOG_YELLOW, "Move color temperature from BLE\n");
        send_move_color();
        break;

    default:
        info_color(LOG_RED, "zigbee app unkown event %d\n", event);
        break;
    }
}
```

Likewise, ble_app_main() is defined to process the request from function “app_request_set()”. The request types are like: start ADV or some specific application.

```
void ble_app_main(app_req_param_t *p_param)
{
    // Link - Peripheral
    app_peripheral_handler(p_param);
}

static void app_peripheral_handler(app_req_param_t *p_param)
{
    ble_err_t status;
    uint8_t host_id;
    ble_profile_info_t *p_profile_info;

    host_id = p_param->host_id;
    p_profile_info = (ble_profile_info_t *)ble_app_link_info[host_id].profile_info;

    switch (p_param->app_req)
    {
    case APP_REQUEST_ADV_START:
        do
        {
            // service data init
            svcs_gatts_data_init(&p_profile_info->svcs_info_gatts.server_info.data);
            svcs_trsps_data_init(&p_profile_info->svcs_info_trsps.server_info.data);

            // set preferred MTU size and data length
            status = ble_cmd_default_mtu_size_set(host_id, DEFAULT_MTU);
            if (status != BLE_ERR_OK)
            {
                info_color(LOG_RED, "ble_cmd_default_mtu_size_set() status = %d\n", status);
                break;
            }

            // enable advertising
            status = adv_init();
            if (status != BLE_ERR_OK)
            {
                info_color(LOG_RED, "adv_init() status = %d\n", status);
                break;
            }

            status = adv_enable(host_id);
            if (status != BLE_ERR_OK)
            {
                info_color(LOG_RED, "adv_enable() status = %d\n", status);
                break;
            }
        } while (0);

        break;

    case APP_REQUEST_TRSPS_DATA_SEND:

        // send data to client
        if ((p_profile_info->svcs_info_trsps.server_info.data.udatni01_cccd & BLEGATT_CCCD_NOTIFICATION) != 0)
        {
            status = ble_svcs_trsps_server_send(  host_id,
                                                  BLEGATT_CCCD_NOTIFICATION,
                                                  p_profile_info->svcs_info_trsps.server_info.handles.hdl
     ……

    default:
        break;
    }
}
```

3.6 zb_event_parse & ble_event_parse

zb_event_parse() is used to parse the event send from zb_app_evt_indication_cb(), likewise, ble_event_parse() use to parse the event send from ble_app_evt_indication_cb()

```
void zb_event_parse(sys_tlv_t *pt_zb_tlv)
{
    switch (pt_zb_tlv->type)
    {
    case ZIGBEE_EVT_TYPE_ZCL_DATA_IDC:
        zigbee_zcl_msg_handler(pt_zb_tlv);
    break;

    default:
        zigbee_evt_handler(pt_zb_tlv);
    break;
    }

    if(pt_zb_tlv)
    {
        vPortFree(pt_zb_tlv);
    }
}
```

```
void ble_event_parse(ble_tlv_t *pt_ble_tlv)
{
    if (pt_ble_tlv != NULL)
    {
        switch (pt_ble_tlv->type)
        {
        case BLE_APP_GENERAL_EVENT:
        {
            ble_evt_param_t *ble_evt = (ble_evt_param_t *)pt_ble_tlv->value;
            ble_evt_handler(ble_evt);
        }
        break;

        case BLE_APP_SERVICE_EVENT:
        {
            ble_evt_att_param_t *p_svcs_param = (ble_evt_att_param_t *)pt_ble_tlv->value;

            switch (p_svcs_param->gatt_role)
            {
            case BLE_GATT_ROLE_CLIENT:
                att_db_link[p_svcs_param->host_id].p_client_db[p_svcs_param->cb_index]->att_handler(p_svcs_param);
                break;

            case BLE_GATT_ROLE_SERVER:
                att_db_link[p_svcs_param->host_id].p_server_db[p_svcs_param->cb_index]->att_handler(p_svcs_param);
                break;

            default:
                break;
            }
        }
        break;

        default:
            break;
        }

        // free
        vPortFree(pt_ble_tlv);
    }
}
```

3.7 ZigBee Event Handle

Implement ZigBee event handler “zigbee_evt_handler ()” in zigbee_evt_handler.c file. This function processes the zigbee stack event. (e.g. network start, device announce)

```
static void _zdo_evt_start(sys_tlv_t *pt_tlv);
static void(*zdo_evt_idc_func_list[])(sys_tlv_t *) = {
    [ZIGBEE_EVT_TYPE_START_IDC - ZIGBEE_EVT_TYPE_START_IDC] = _zdo_evt_start,
    [ZIGBEE_EVT_TYPE_DEVICE_ANNCE_IDC - ZIGBEE_EVT_TYPE_START_IDC] = NULL,
    [ZIGBEE_EVT_TYPE_LEAVE_IDC - ZIGBEE_EVT_TYPE_START_IDC] = NULL,
    [ZIGBEE_EVT_TYPE_DEVICE_ASSOCIATED_IDC - ZIGBEE_EVT_TYPE_START_IDC] = NULL,
    [ZIGBEE_EVT_TYPE_PANID_CONFLICT_IDC - ZIGBEE_EVT_TYPE_START_IDC] = NULL,
};

static void _zdo_evt_start(sys_tlv_t *pt_tlv)
{
    zigbee_nwk_start_idc_t *pt_start_idc = (zigbee_nwk_start_idc_t*)pt_tlv->value;
    if(pt_start_idc->status != 0)
    {
        info_color(LOG_RED, "Device do rejoin\n");
        zigbee_app_evt_change(APP_NOT_JOINED_EVT);
    }
    else
    {
        info_color(LOG_GREEN, "Device join success\n");
        info_color(LOG_GREEN, "PAN: %04X, ShortAddr: %04X,
              MAC: %02X:%02X:%02X:%02X:%02X:%02X:%02X:%02X\n",
               pt_start_idc->panID, pt_start_idc->nwkAddr,
               pt_start_idc->ieee_addr[7], pt_start_idc->ieee_addr[6],
               pt_start_idc->ieee_addr[5], pt_start_idc->ieee_addr[4],
               pt_start_idc->ieee_addr[3], pt_start_idc->ieee_addr[2],
               pt_start_idc->ieee_addr[1], pt_start_idc->ieee_addr[0]);
    }
}
void zigbee_evt_handler(sys_tlv_t *pt_tlv)
{
    if((pt_tlv->type >= ZIGBEE_EVT_TYPE_ZDO_START_IDC) &&
       (pt_tlv->type <= ZIGBEE_EVT_TYPE_ZDO_FINISH_IDC))
    {
        if(zdo_evt_idc_func_list[pt_tlv->type - ZIGBEE_EVT_TYPE_START_IDC])
            zdo_evt_idc_func_list[pt_tlv->type - ZIGBEE_EVT_TYPE_START_IDC](pt_tlv);
    }
}
```

3.8 ZCL Message Handle

Implement ZigBee ZCL message handler “zigbee_zcl_msg_handler ()” in zigbee_zcl_msg_handler.c file. This function processes the ZCL messages.

```
void zigbee_zcl_msg_handler(sys_tlv_t *pt_tlv)
{
    zigbee_zcl_data_idc_t *pt_zcl_msg = (zigbee_zcl_data_idc_t *)pt_tlv->value;
    do
    {
        if(!pt_zcl_msg)
            break;
        info("Recv ZCL message\n");
        info("Cluster %04x, cmd %d\n", pt_zcl_msg->clusterID, pt_zcl_msg->cmd);
        util_log_mem(UTIL_LOG_INFO, "  ", (uint8_t *)pt_zcl_msg->cmdFormat, pt_zcl_msg->cmdFormatLen, 0);
    } while(0);
}
```

3.9 BLE Event Handle

Implement BLE event handler “ble_evt_handler ()” in ble_app.c file. This function processes the BLE host stack event. (e.g. ADV enable result and connection complete)

```
static void ble_evt_handler(ble_evt_param_t *p_param)
{
    switch (p_param->event)
    {
    case BLE_ADV_EVT_SET_ENABLE:
    {
        ble_evt_adv_set_adv_enable_t *p_adv_enable = (ble_evt_adv_set_adv_enable_t *)&p_param-
>event_param.ble_evt_adv.param.evt_set_adv_enable;

        if (p_adv_enable->status == BLE_HCI_ERR_CODE_SUCCESS)
        {
            if (p_adv_enable->adv_enabled == true)
            {
                if (g_advertising_host_id != BLE_HOSTID_RESERVED)
                {
                    ble_app_link_info[g_advertising_host_id].state = STATE_ADVERTISING;
                }
                info_color(LOG_GREEN, "Advertising...\n");
            }
            else
            {
                if (g_advertising_host_id != BLE_HOSTID_RESERVED)
                {
                    ble_app_link_info[g_advertising_host_id].state = STATE_STANDBY;
                }
                info_color(LOG_GREEN, "Idle.\n");
            }
        }
        else
        {
            info_color(LOG_RED, "Advertising enable failed.\n");
        }
    }
    break;

    case BLE_GAP_EVT_CONN_COMPLETE:
    {
        ble_evt_gap_conn_complete_t *p_conn_param = (ble_evt_gap_conn_complete_t *)&p_param-
>event_param.ble_evt_gap.param.evt_conn_complete;

        if (p_conn_param->status != BLE_HCI_ERR_CODE_SUCCESS)
        {
            info_color(LOG_RED, "Connect failed, error code = 0x%02x\n", p_conn_param->status);
        }
        else
        {
            ble_app_link_info[p_conn_param->host_id].state = STATE_CONNECTED;
            info_color(LOG_GREEN, "Connected, ID=%d, Connected to %02x:%02x:%02x:%02x:%02x:%02x\n",
                       p_conn_param->host_id,
                       p_conn_param->peer_addr.addr[5],
                       p_conn_param->peer_addr.addr[4],
                       p_conn_param->peer_addr.addr[3],
                       p_conn_param->peer_addr.addr[2],
                       p_conn_param->peer_addr.addr[1],
                       p_conn_param->peer_addr.addr[0]);
        }
    }
    break;

    ……

    default:
        break;
    }
}
```

3.10 Event Message Indication

Implement callback function of event message indication “zb_app_evt_indication_cb()” / “ble_app_evt_indication_cb()” in zigbee_app.c/ble_app.c file. This function sendd event message to application task.

```
void zb_app_evt_indication_cb(uint32_t data_len)
{
    int i32_err;
    uint8_t *pBuf = pvPortMalloc(data_len);
    multi_app_queue_t t_app_q;
    do
    {
        if(!pBuf)
            break;
        t_app_q.event = 0;
        i32_err = zigbee_event_msg_recvfrom(pBuf, &data_len);
        t_app_q.param_type = QUEUE_TYPE_ZIGBEE;
        t_app_q.param.pt_zb_tlv = (sys_tlv_t *)pBuf;
        if (i32_err == 0)
        {
            while (xQueueSendToBack(app_msg_q, &t_app_q, 20) != pdTRUE);
        }
        else
        {
            info_color(LOG_RED, "[%s] sys_err = %d !\n", __func__, i32_err);
            vPortFree(pBuf);
        }
    } while (0);
}
```

```
void ble_app_evt_indication_cb(uint32_t data_len)
{
    int i32_err;
    uint8_t *p_buf = pvPortMalloc(data_len);
    multi_app_queue_t p_app_q;
    do
    {
        if (!p_buf)
        {
            break;
        }

        p_app_q.event = 0;
        i32_err = ble_event_msg_recvfrom(p_buf, &data_len);
        p_app_q.param_type = QUEUE_TYPE_BLE;
        p_app_q.param.pt_ble_tlv = (ble_tlv_t *)p_buf;

        if (i32_err == 0)
        {
            while (xQueueSendToBack(app_msg_q, &p_app_q, 20) != pdTRUE);
        }
        else
        {
            info_color(LOG_RED, "[%s] err = %d !\n", __func__, (ble_err_t)i32_err);
            vPortFree(p_buf);
        }
    } while (0);
}
```

4. Multi-Protocol Demonstration

The RT58x ZigBee/BLE SDK provides tools and instructions to let you develop your own applications. Refer to the document "*RT58x_Zigbee_Application_Guide*" section 4.1 ZigBee Demonstration to know how to start coordinator and let end device and router to join ZigBee network.

4.1 Multi-Protocol Project

Multi-protocol demo project concurrent runs two protocols, ZigBee and BLE. For ZigBee and BLE application, we use BLE peripheral device with different ZigBee role end device, router and coordinator to develop following multi-protocol demonstration:

ZC_Gateway_Module_BLE_TRSP: base on project ZigBee gateway module and send gateway command over BLE transparent service instead of UART.

ZED_Switch_BLE_TRSP: base on project ZigBee switch and send specific data string over BLE transparent service to emulate the button of switch triggered.

ZED_Switch_BLE_TRSP_FOTA: base on project ZED_Switch_BLE_TRSP and the FW can be update over BLE.

ZR_Light_BLE_TRSP: base on project ZigBee light and send specific data string over BLE transparent service to notify ZR reset to factory new.

4.2 ZigBee Coordinator gateway module

The initial procedure is same as project gateway module of ZigBee demo.

4.3 ZigBee End Device Switch

The switch joining/setting procedure is same as project Switch of ZigBee demo.

4.4 ZigBee Router light

The light joining/setting procedure is same as project Light of ZigBee demo.

4.5 BLE Transparent Service

Transparent Service (TRSP Service) is a proprietary customer defined service which uses to

emulate the serial port over BLE. About the procedure of creating a customer defined service, please see "*RT58x_BLE_SDK_Service_Profile_Guide.pdf*" for more details. "ble_service_trsps.c/ ble_service_trsps.h" are the implementation files.

Service	UUID	Description
TRSP Service	0x00112233445566778899AABBCCDDEEFF	Serial data transferred via BLE.
Characteristic	UUID	Description
Read	0x101112131415161718191A1B1C1D1E1F	The client can get the static data from this characteristic.
Notify	0x303132333435363738393A3B3C3D3E3F	The peer device can send data to client by writing this characteristic.
Read/ Write	0x505152535455565758595A5B5C5D5E5F	The peer device can send data to the client or receive data from the client via this characteristic.

4.6 Handled Received Data over BLE

Implement the callback function "ble_svcs_trsps_evt_handler" which is initialized in "server_profile_init" function to handle the data received over BLE.

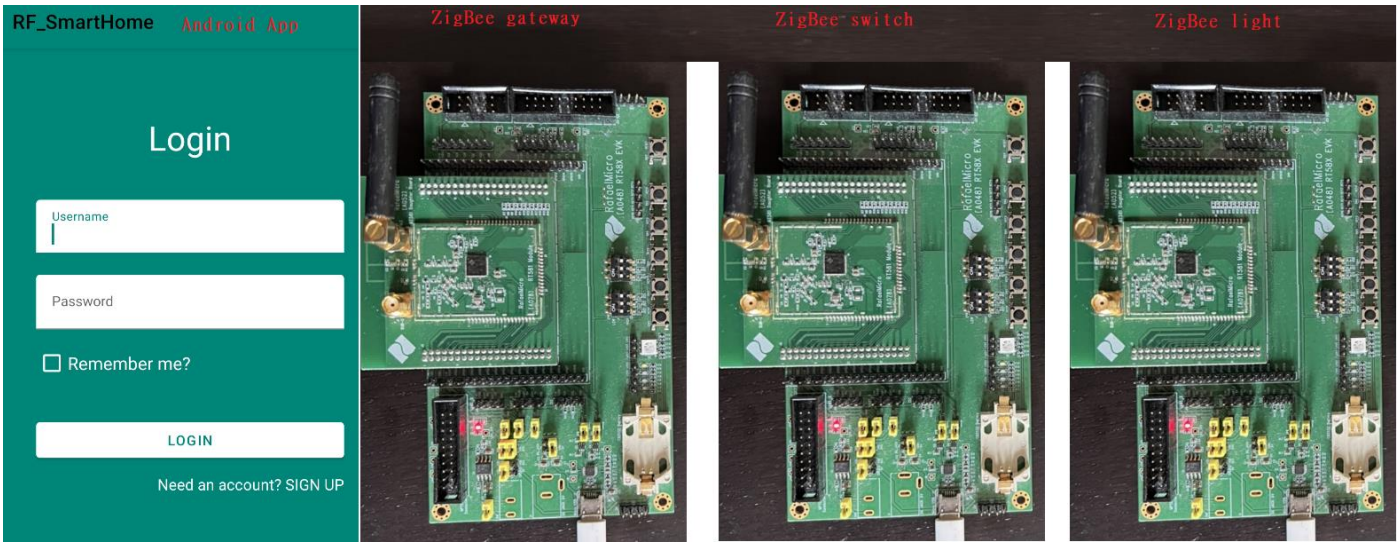
```
static ble_err_t server_profile_init(uint8_t host_id)
{
    ble_err_t status = BLE_ERR_OK;
    ble_profile_info_t *p_profile_info = (ble_profile_info_t *)ble_app_link_info[host_id].profile_info;

    // set link's state
    ble_app_link_info[host_id].state = STATE_STANDBY;

    do
    {
        // GAP Related
        // -------------------------------------
        status = ble_svcs_gaps_init(host_id, BLE_GATT_ROLE_SERVER, &(p_profile_info->svcs_info_gaps), NULL);
        if (status != BLE_ERR_OK)
        {
            break;
        }

        // set GAP device name
        status = ble_svcs_gaps_device_name_set((uint8_t *)DEVICE_NAME_STR, sizeof(DEVICE_NAME_STR));
        if (status != BLE_ERR_OK)
        {
            break;
        }

        // GATT Related
        // -------------------------------------
        status = ble_svcs_gatts_init(host_id, BLE_GATT_ROLE_SERVER, &(p_profile_info->svcs_info_gatts), NULL);
        if (status != BLE_ERR_OK)
        {
            break;
        }

        // DIS Related
        // -------------------------------------
        status = ble_svcs_dis_init(host_id, BLE_GATT_ROLE_SERVER, &(p_profile_info->svcs_info_dis), NULL);
        if (status != BLE_ERR_OK)
        {
            break;
        }

        // TRSPS Related
        // -------------------------------------
        status = ble_svcs_trsps_init(host_id, BLE_GATT_ROLE_SERVER, &(p_profile_info->svcs_info_trsps),
                                     ble_svcs_trsps_evt_handler);
        if (status != BLE_ERR_OK)
        {
            break;
        }
    } while (0);

    return status;
}
```

```
static void ble_svcs_trsps_evt_handler(ble_evt_att_param_t *p_param)
{
    if (p_param->gatt_role == BLE_GATT_ROLE_SERVER)
    {
        /* ----------------- Handle event from client ----------------- */
        switch (p_param->event)
        {
        case BLESERVICE_TRSPS_UDATRW01_WRITE_EVENT:
        case BLESERVICE_TRSPS_UDATRW01_WRITE_WITHOUT_RSP_EVENT:
            if (strncmp((char *)p_param->data, BUTTON_1_STR, sizeof(BUTTON_1_STR)-1) == 0)
            {
                zigbee_app_evt_change(APP_LED_TOGGLE_EVT, false);
            }
            else if (strncmp((char *)p_param->data, BUTTON_2_STR, sizeof(BUTTON_2_STR)-1) == 0)
            {
                zigbee_app_evt_change(APP_LED_LEVEL_UP_EVT, false);
            }
            else if (strncmp((char *)p_param->data, BUTTON_3_STR, sizeof(BUTTON_3_STR)-1) == 0)
            {
                zigbee_app_evt_change(APP_LED_LEVEL_DOWN_EVT, false);
            }
            else if (strncmp((char *)p_param->data, BUTTON_4_STR, sizeof(BUTTON_4_STR)-1) == 0)
            {
                zigbee_app_evt_change(APP_LED_MOVE_TEMP_EVT, false);
            }
            p_param->data[p_param->length] = '\0';
            info_color(LOG_CYAN, "%s\n", p_param->data);
            break;

        case BLESERVICE_TRSPS_UDATRW01_READ_EVENT:
        {
            ble_err_t status;
            const uint8_t readData[] = "UDATRW01 data";
            ble_gatt_data_param_t gatt_data_param;

            gatt_data_param.host_id = p_param->host_id;
            gatt_data_param.handle_num = p_param->handle_num;
            gatt_data_param.length = SIZE_STRING(readData);
            gatt_data_param.p_data = (uint8_t *)readData;

            status = ble_svcs_data_send(TYPE_BLE_GATT_READ_RSP, &gatt_data_param);;
            if (status != BLE_ERR_OK)
            {
                info_color(LOG_RED, "ble_gatt_read_rsp status: %d\n", status);
            }
        }
        break;

        default:
            break;
        }
    }
}
```

4.7 Running Demo ZC_Gateway_Module_BLE_TRSP

This chapter introduces the operation method of Rafael Smart Home App. Users can use BLE communication through Smart Home App and Gateway module to achieve the purpose of operating ZigBee network.

4.7.1 Set up the environment:



The SDK application path is as follows:

Zigbee Gateway FW:

..\Project\Application\Multi_Protocol_Demo\ZC_Gateway_Module_BLE_TRSP

Zigbee Light FW:

..\Project\Application\Zigbee_Demo\Light

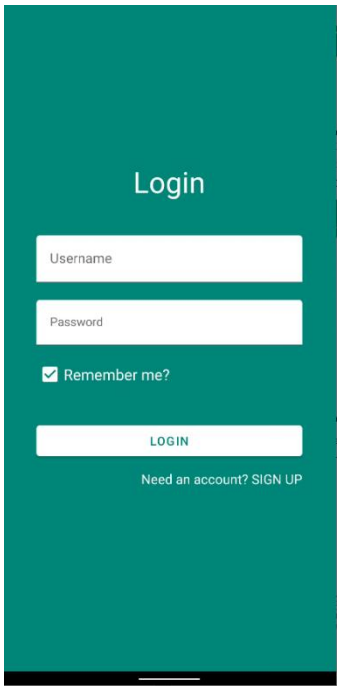
Zigbee Switch FW:

..\Project\Application\Zigbee_Demo\Switch

4.7.2 APP User Interface

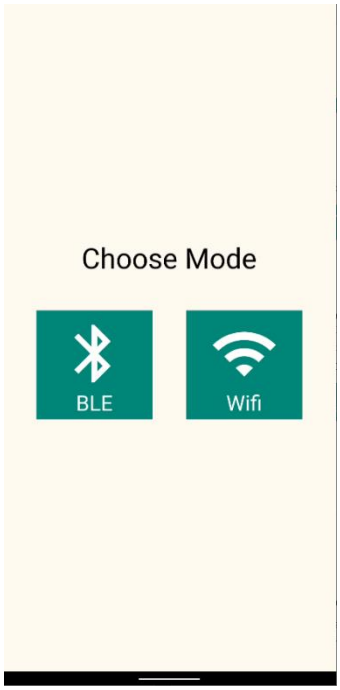
- Login

  This page can obtain private settings by entering the user name and password, or the user can directly press the LOGIN button without entering the account password.



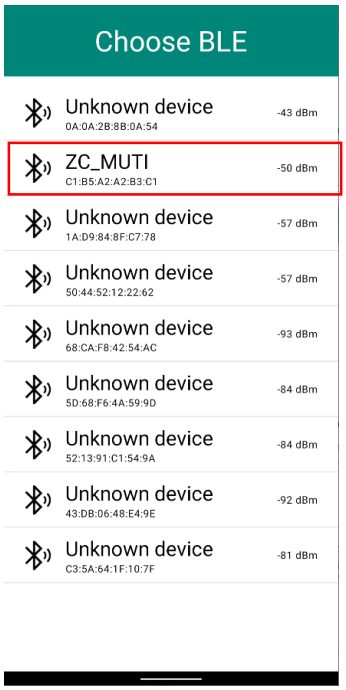
- Choose BLE or WIFI mode.

  This page provides users to select BLE or WIFI operation mode. This chapter mainly describes the BLE mode operation.



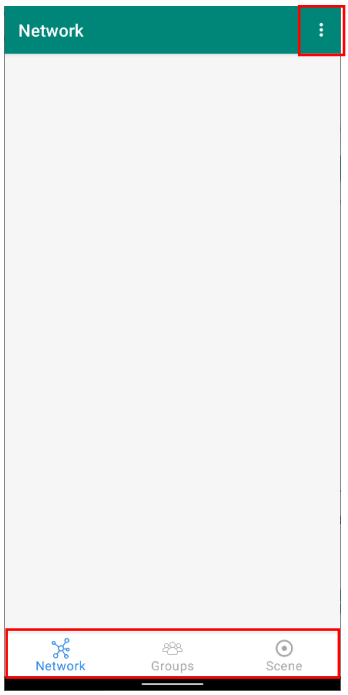
- BLE connection page

  This page scans all connectable BLE devices, and the user must select the Gateway module device to be connected.



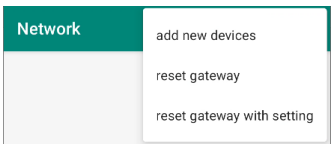
➢ Main user interface

The main user interface consists of several operation blocks: Menu, Network and Groups.



- Menu:

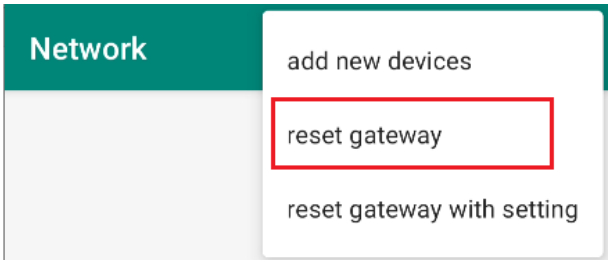
  Contains operations such as new devices, reset gateway and reset gateway with setting.



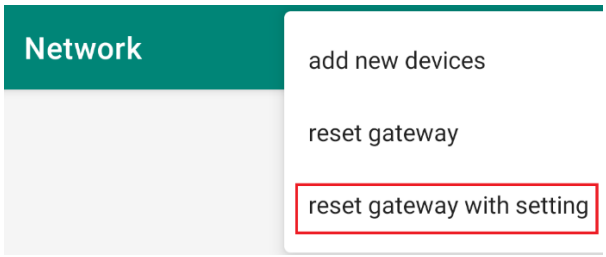
- Network:
  This is the device management page, all devices that have joined the network will be displayed on this page.
- Groups:
  The group operation page can manage multiple devices at the same time.
- Scene:
  The scene operation page can manage multiple devices at the same time, and further operate each device to different states according to the needs.

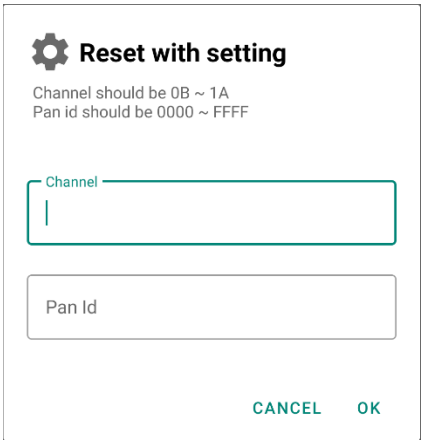
➢ **Pre-actions for connecting to the gateway for the first time.**

When the user connects to the gateway with the smart home app for the first time, the user must first press "Reset gateway" on the menu to reset the gateway, and then reconnect to the gateway. The purpose is setting the channel and PAN ID of ZigBee networking and also to prevent old data from being stored in the gateway device. We clear it to avoid unexpected data causing erroneous behavior.



After pressing “Reset gateway”, the Rafael Smart Home App will give a default channel and PAN ID for the gateway. If users want to set a specific channel (for avoid interference) or PAN ID (for easier identification), please press “Reset gateway with setting” on the menu to set channel and PAN ID.





4.7.3 Operation steps:

The following will describe the operation steps for an LED Device:

[Reminder] The pre-actions for connecting to the gateway for the first time must be performed.

- Step 1

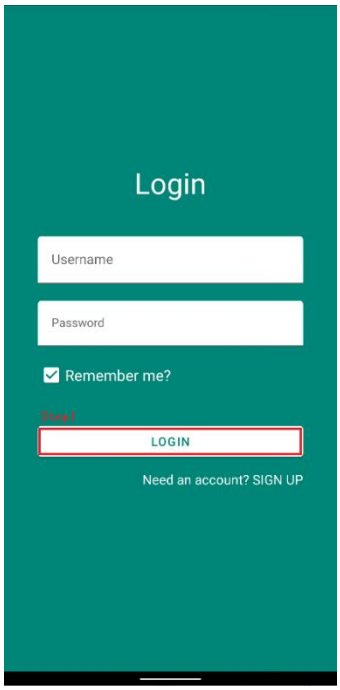
  Open Smart Home APP and press LOGIN button.

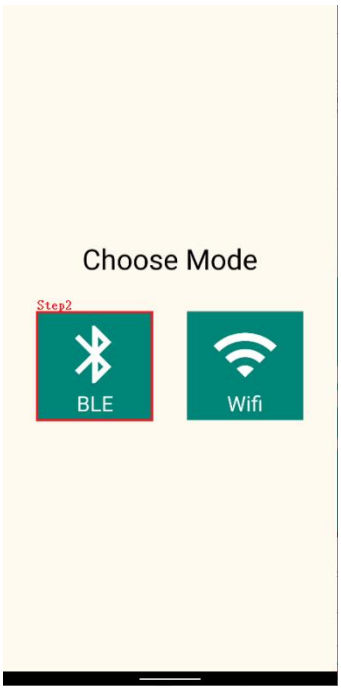
- Step 2

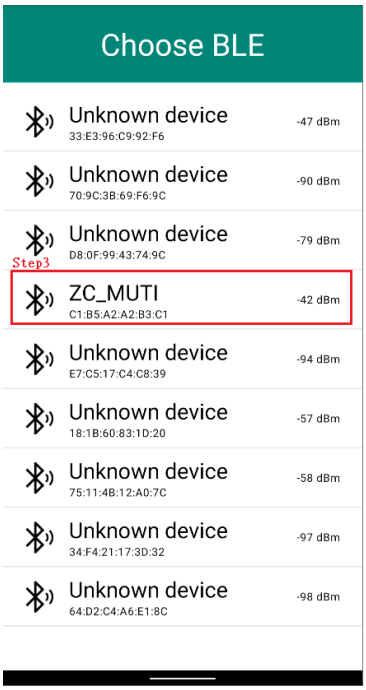
  Select BLE mode.

- Step 3

  Find the gateway module and connect it.







- Step 4

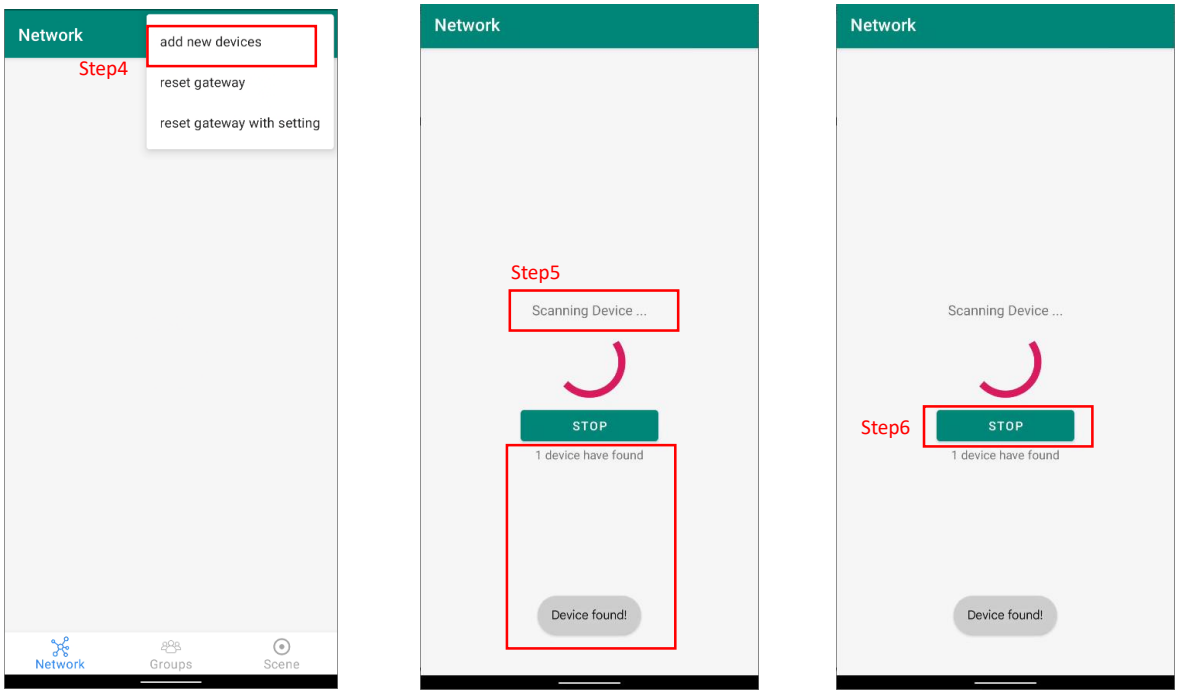
  [Menu] Press add new devices.

- Step 5

  Go to the Scanning device page and wait for the all device to join the network.

- Step 6

  Press the “STOP” button to return to the Network page, which will display the devices connected to the network.



- Step7

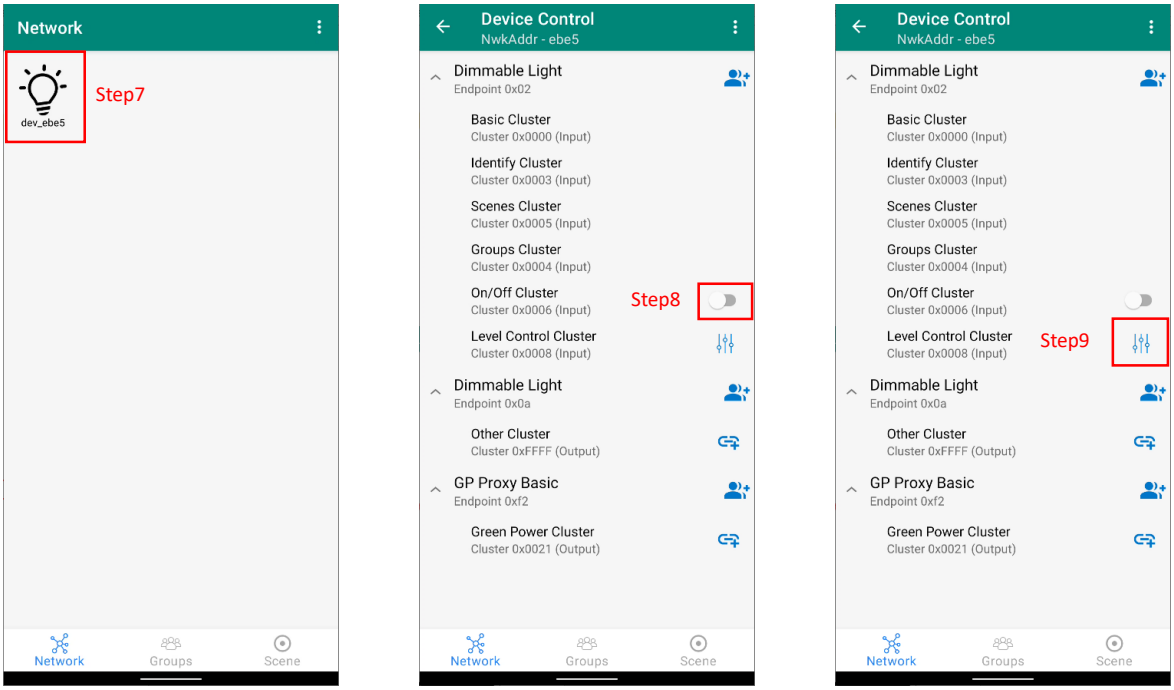
  Press the device icon to enter the device information and operation page.

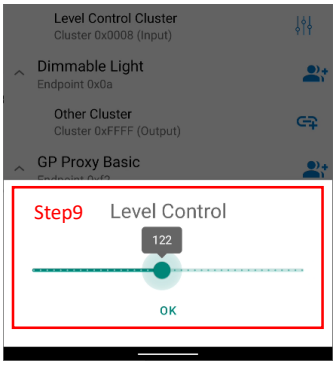
- Step8

  Operation LED ON/OFF.

- Step9

  Operation LED Level [Range from 0 to 255].





4.7.4 Bind operation steps

The following will describe the LED and Switch devices into the ZigBee network and the bind operation steps:

[Reminder] The pre-actions for connecting to the gateway for the first time must be performed.

- Step 1

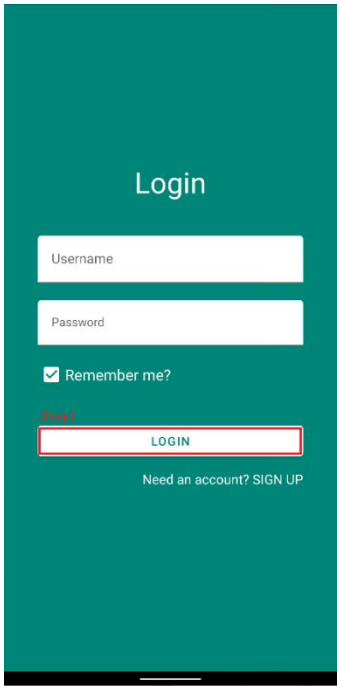
  Open Smart Home APP and press LOGIN button.

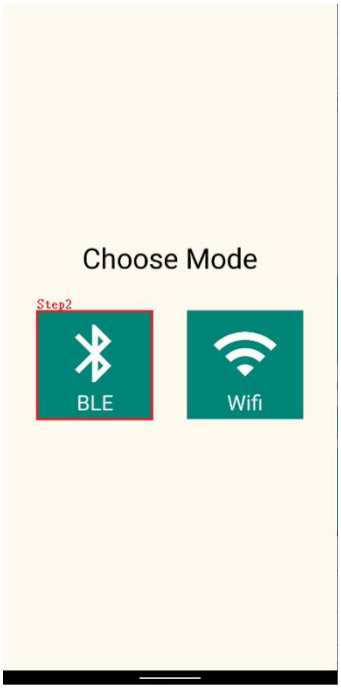
- Step 2

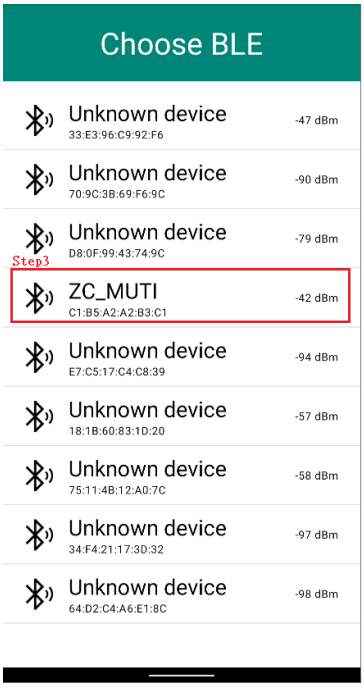
  Select BLE mode.

- Step 3

  Find the gateway module and connect it.







- Step 4

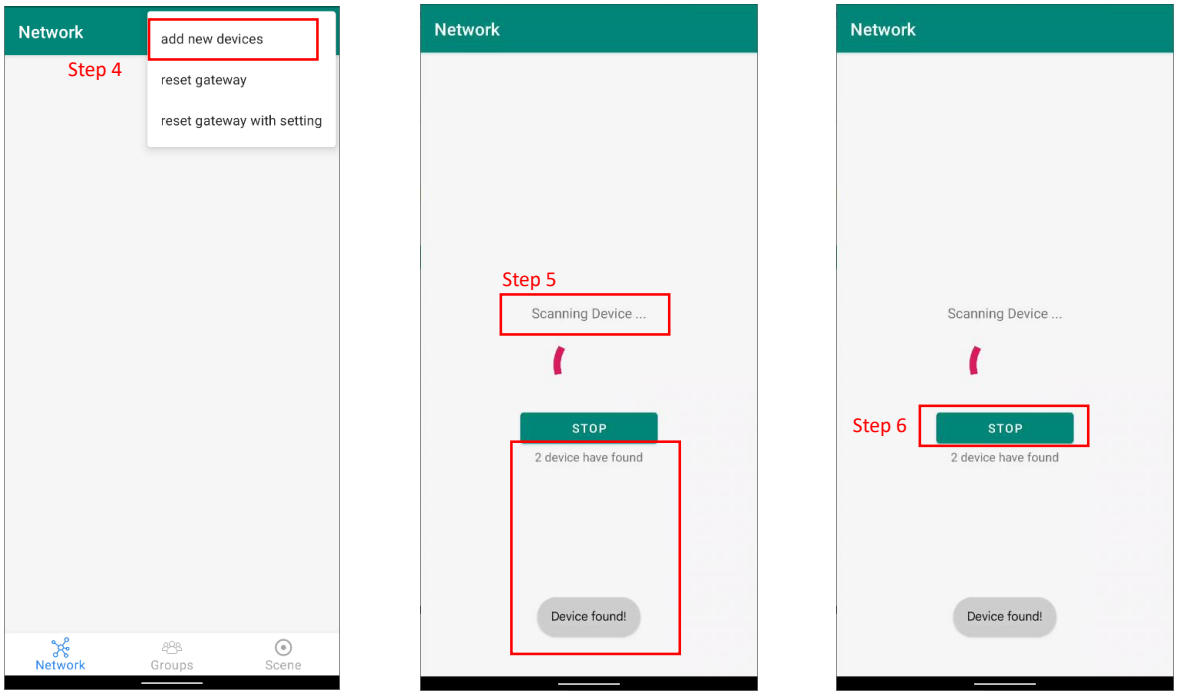
  [Menu] Press add new devices.

- Step 5

Go to the Scanning device page and wait for the all device to join the network.

- Step 6

  Press the “STOP” button to return to the Network page, which will display the devices connected to the network.



- Step 7

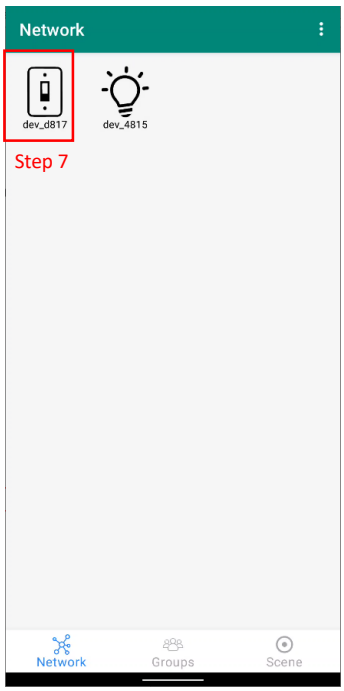
  Press the switch device icon to enter the device information and operation page.

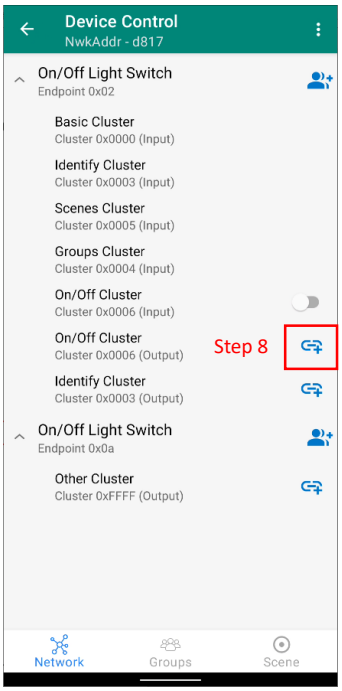
- Step 8

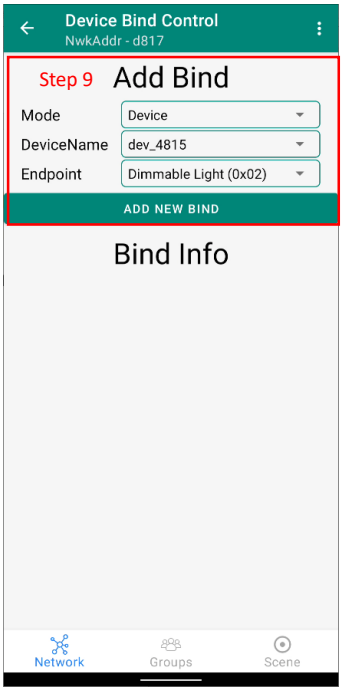
  Press the bind icon to enter the operation page.

- Step 9

  Select the device (or group) and Endpoint that you want to bind, and then press the “ADD NEW BIND” button.

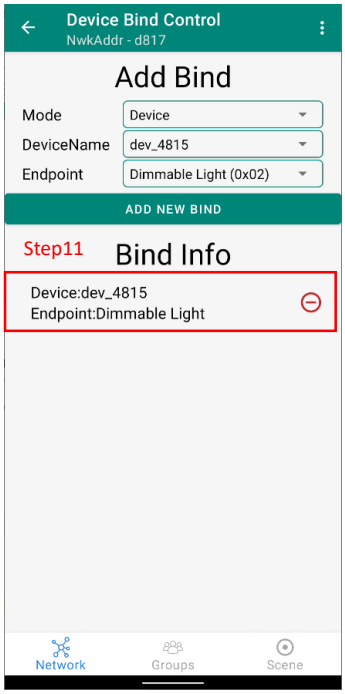




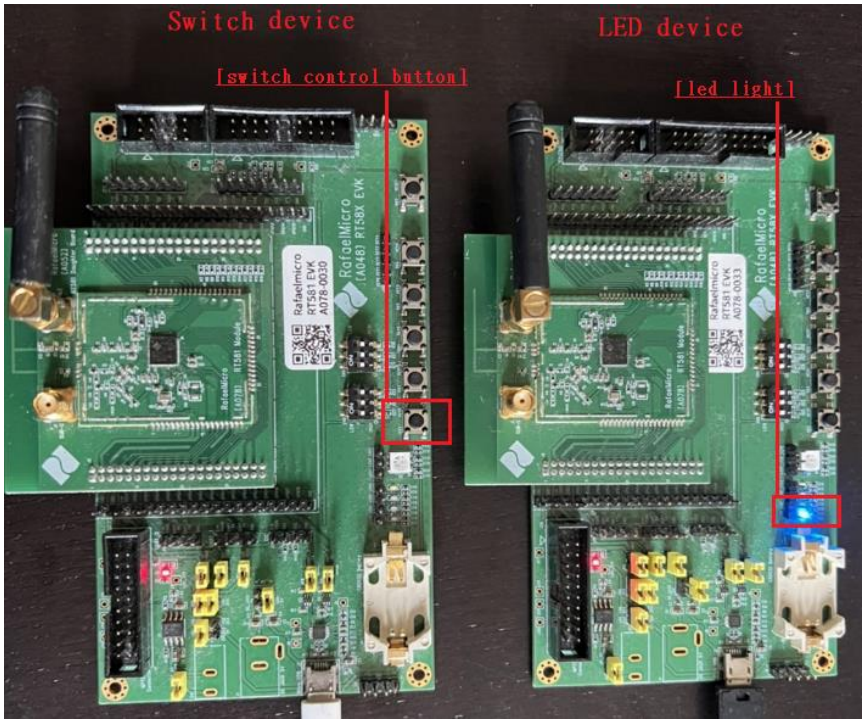


- Step11

  When the bind is successful, a prompt message will pop up on the UI, and Bind Info will be displayed.



- Now, you can use the switch button to control the LED lights.



4.7.5 Group operation steps

The following operations allow four devices to join the network and join the same group.

[Reminder] The pre-actions for connecting to the gateway for the first time must be performed.

- Step 1

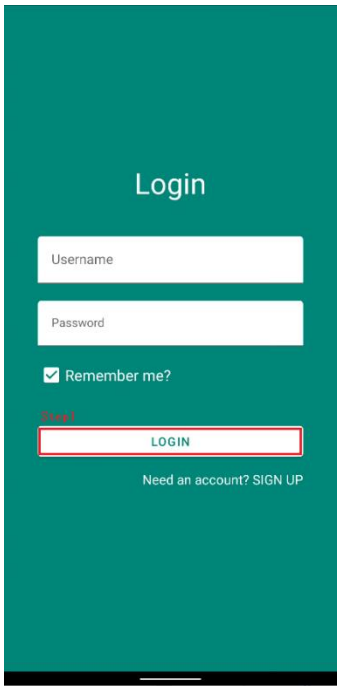
  Open Smart Home APP and press LOGIN button.

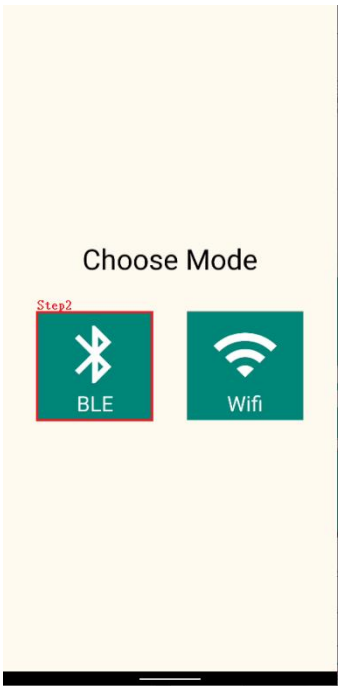
- Step 2

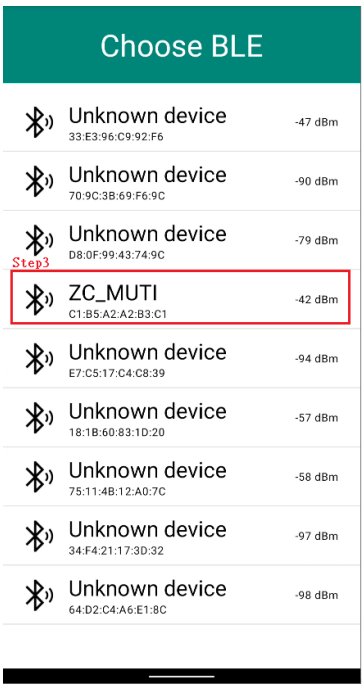
  Select BLE mode.

- Step 3

  Find the gateway module and connect it.





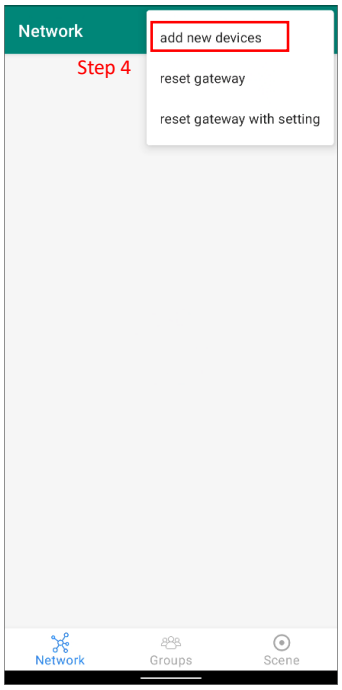


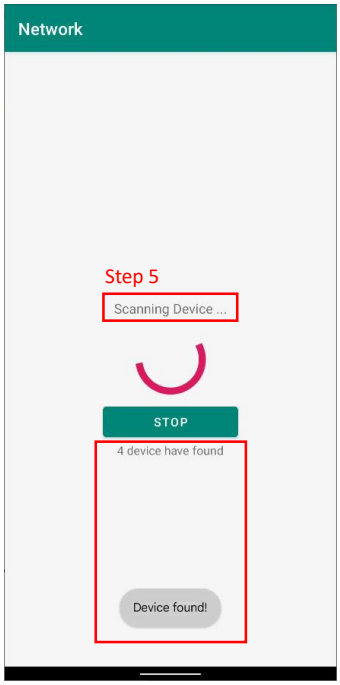
- Step 4

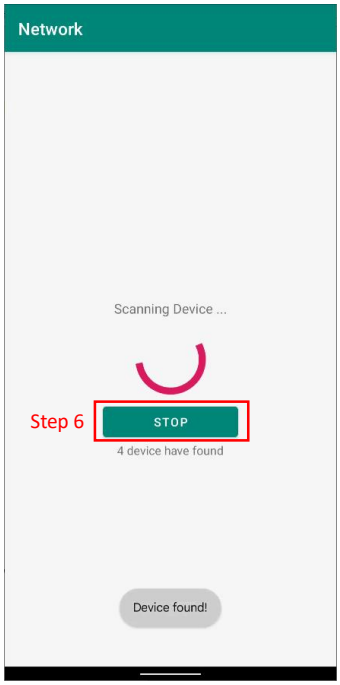
  [Menu] Press add new devices.
- Step 5

  Go to the Scanning device page and wait for the all device to join the network.
- Step 6

  Press the “STOP” button to return to the Network page, which will display the devices connected to the network.







- Step 7

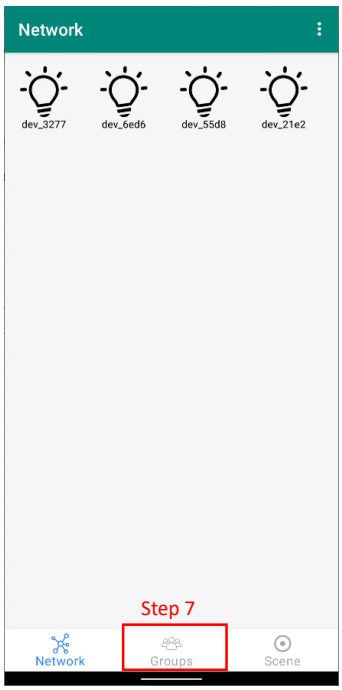
  Press the "Group" button to enter the management page.

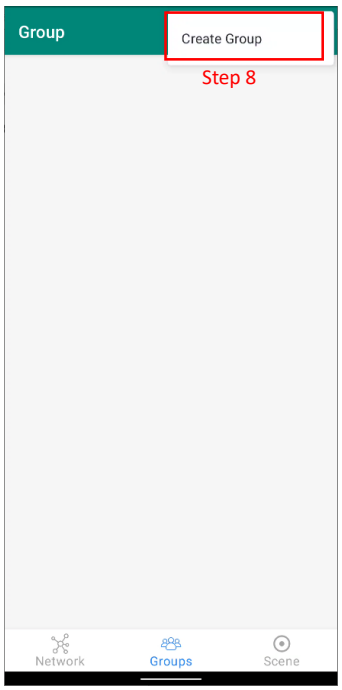
- Step 8

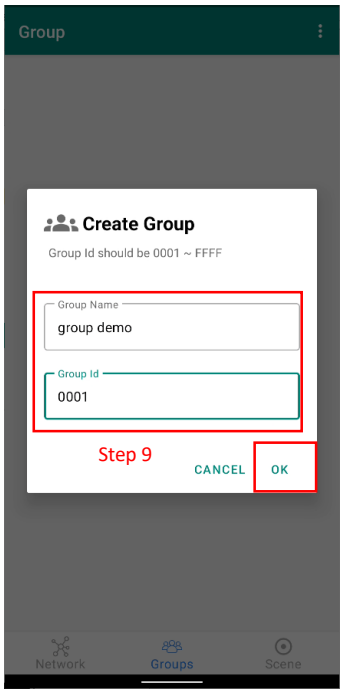
  Press the "Create Group" button on the menu.

- Step 9

  Set the Group information and press the “OK” button.







- Step 10

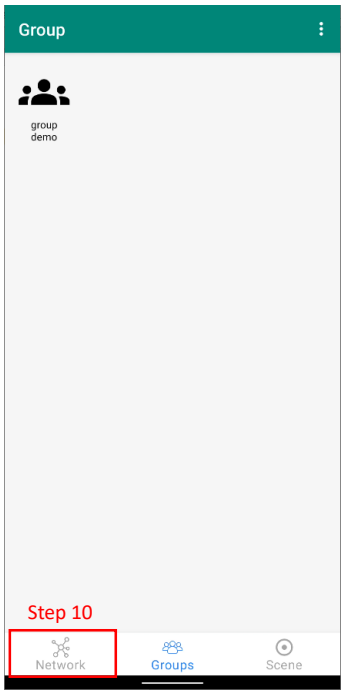
The Group is created successfully and returns to the “Network” page.

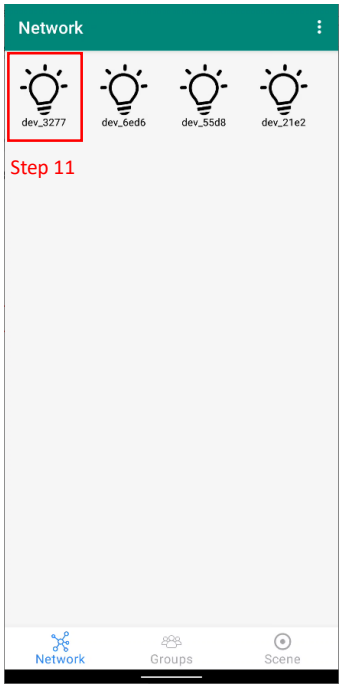
- Step 11

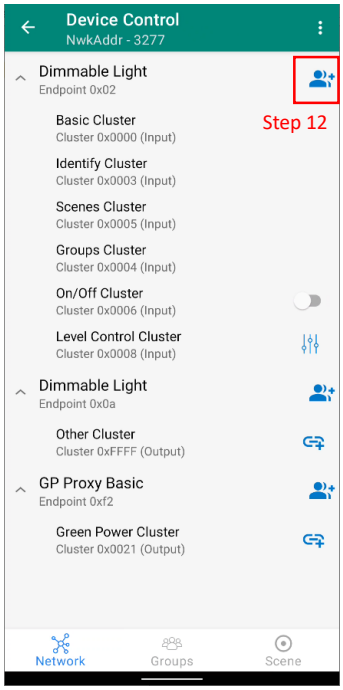
  Click Device 1 to enter the device operation page.

- Step 12

  To add this device to a group, press +Group Icon as shown.







- Step 13

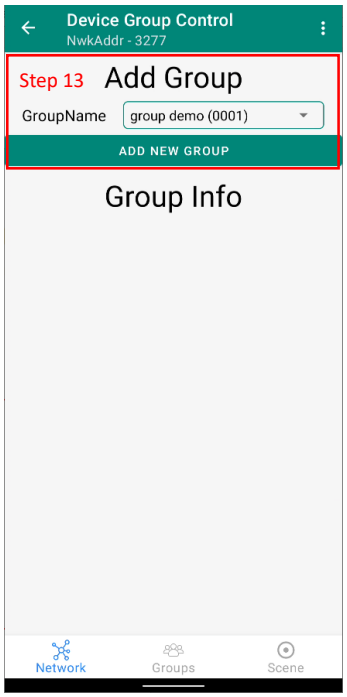
  Select the group you want to join and press the "ADD NEW GROUP" button.

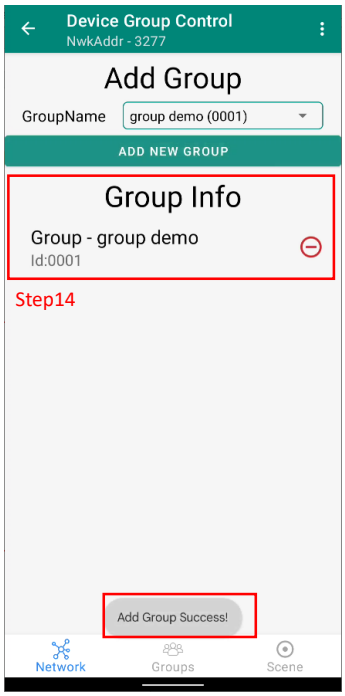
- Step 14

  After successfully joining the group, the UI will display the group information to Group Info.

- Step 15

  And so on, please follow steps 12 to 15 to join the other two devices to the group.





- Step 16

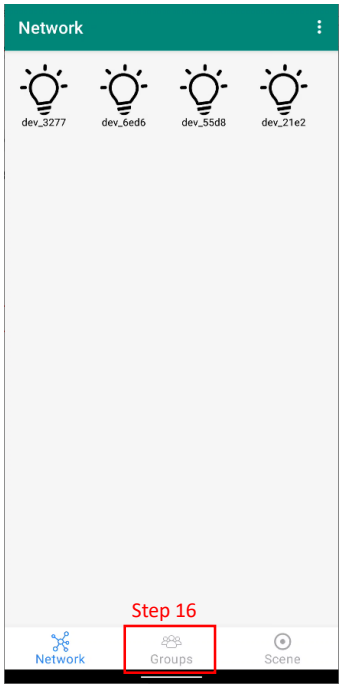
  Now go back to the Network page, and then go to the Group management page.

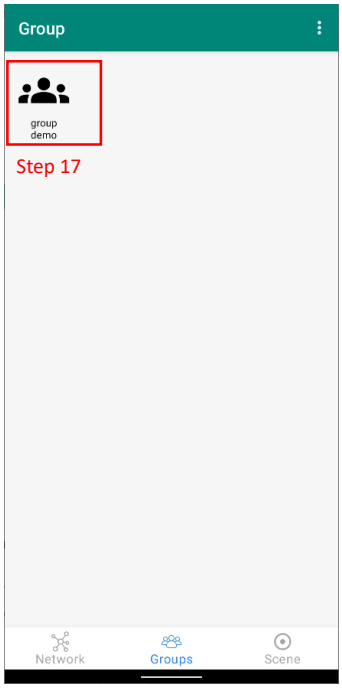
- Step 17

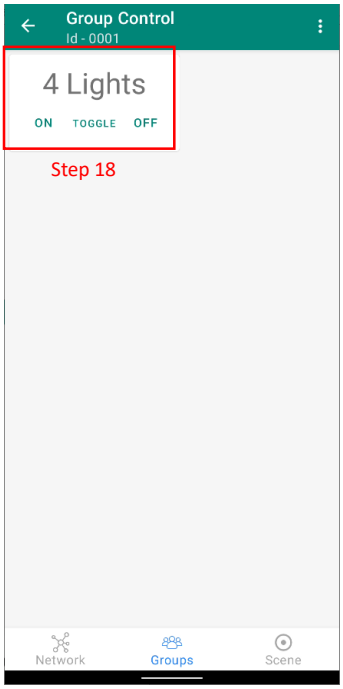
  Press the “group demo” icon to enter the operation page.

- Step 18

  You can now act on this group.







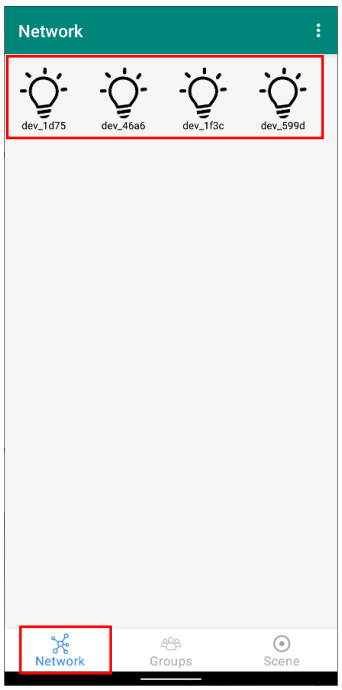
4.7.6 Scene operation steps

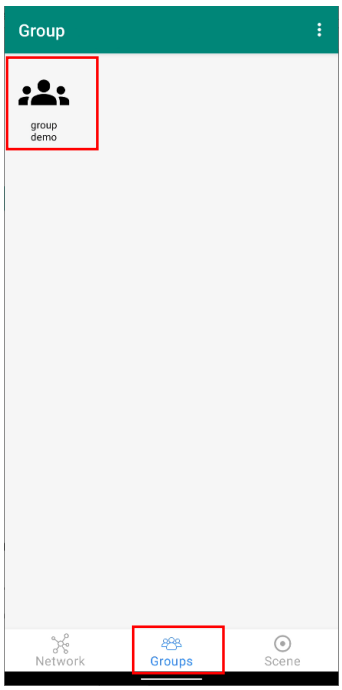
The following operations allow four devices to join the network and join the same group (Implement section 4.7.5 first), then use the scene operation to store the state of the specified group.

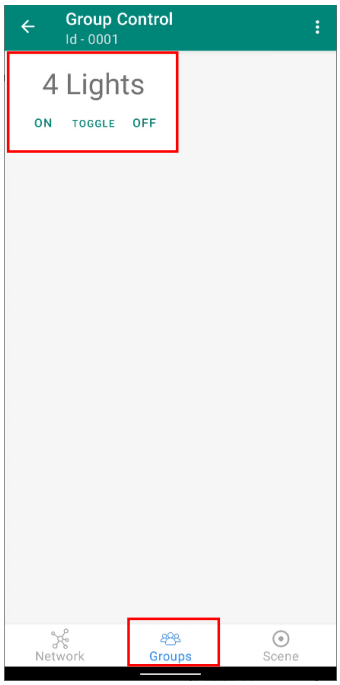
[Reminder] The pre-actions for connecting to the gateway for the first time must be performed.

- Step 1

  Please implement chapter 4.7.5 first to create a group of 4 devices. The operator must have the same picture below, and the function can be operated correctly.







- Step 2

  Set this group of devices to the desired state for the user. For example: users can use Group to set the power of all devices to ON or OFF, and can also individually control the devices in the group.

- Step 3

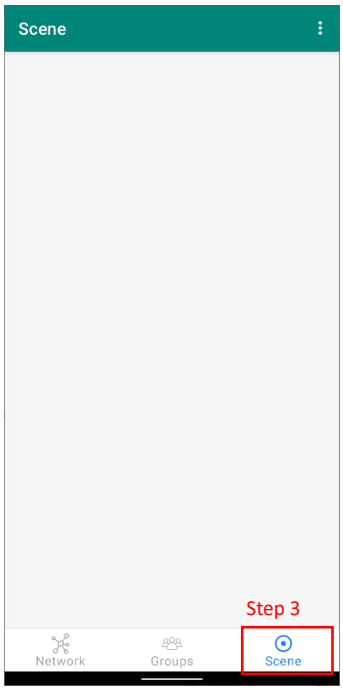
  Press the “Scene” page.

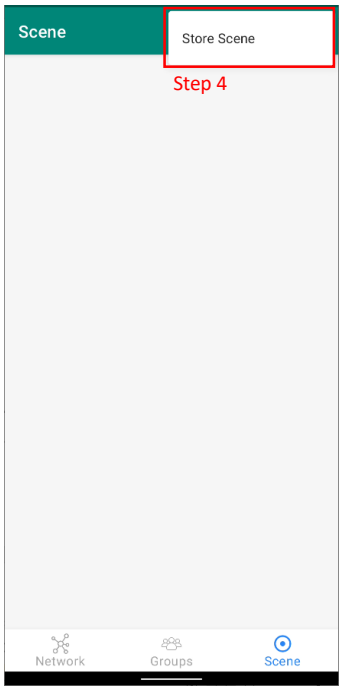
- Step 4

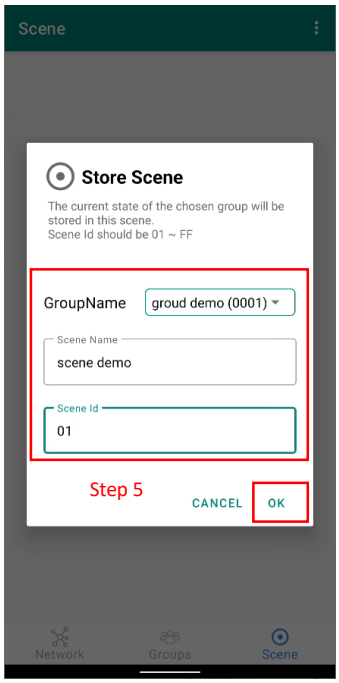
  Press the "Store Scene" button on the menu.

- Step 5

  Select the group name, set the scene name and id, and press the "OK" button, this Scene will save the state of the group settings in step 2.

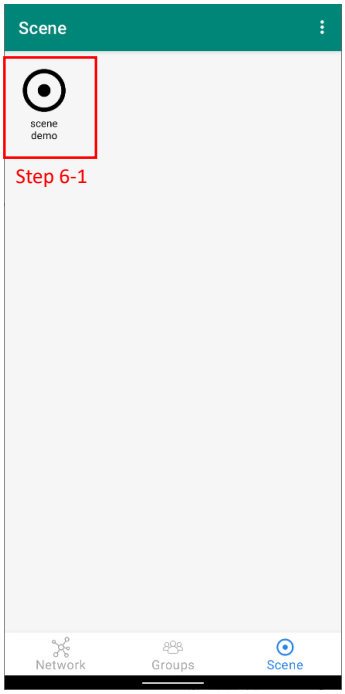


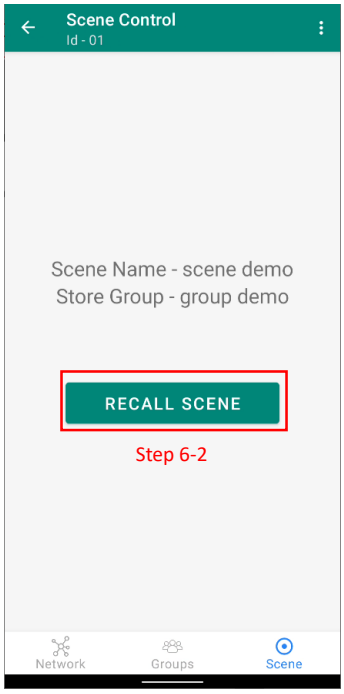




➢ Step 6

The UI will create a scene button, press this button (6-1) to enter the operation UI, then press the “RECALL SCENE” button (6-2) to return to the group state recorded in step 2





4.7.7 Other operation

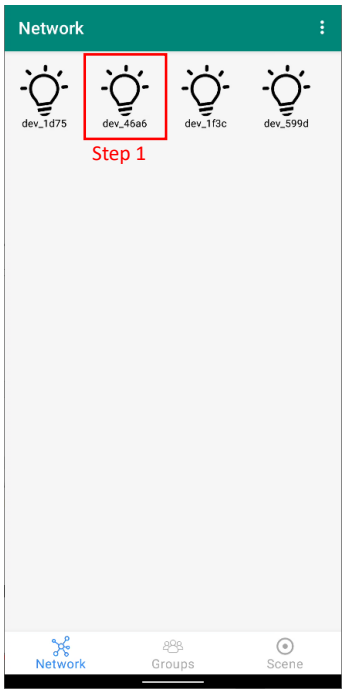
- Remove device

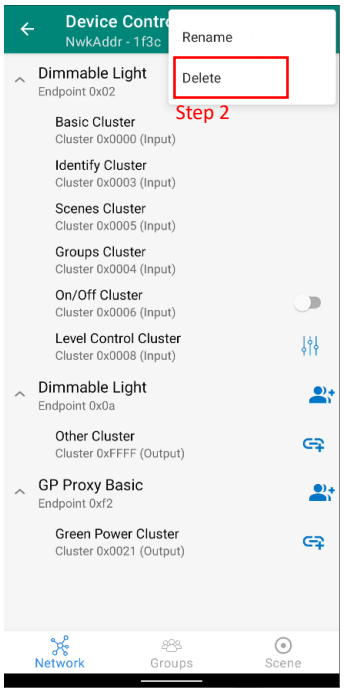
➢ Step 1

Select the device you want to remove, press the icon and enter the operation page.

➢ Step 2

Press the “Delete” button on the Menu.





- Rename device

➢ Step 1

Select the device you want to rename, press the icon and enter the operation page.

➢ Step 2

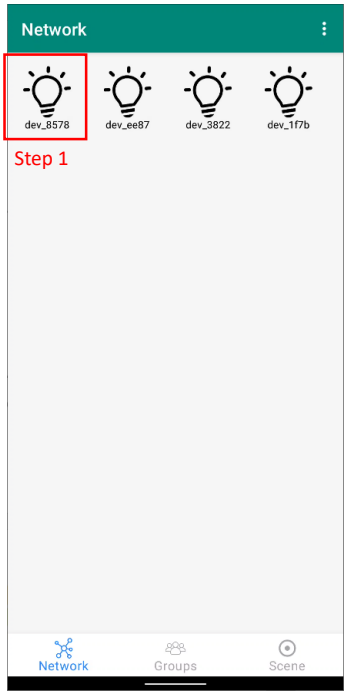
Press the “Rename” button on the Menu.

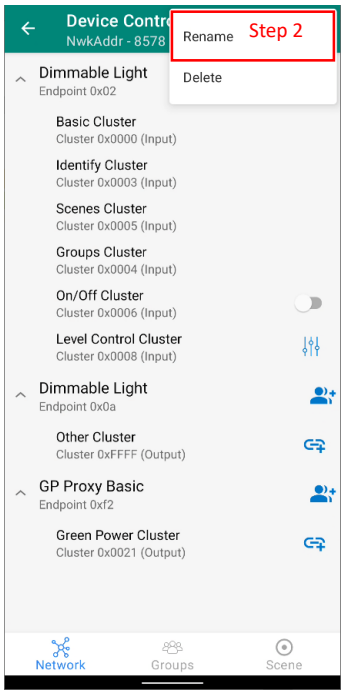
➢ Step 3

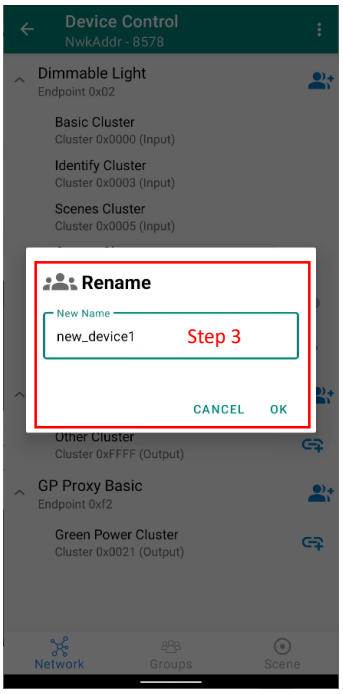
Set a new name and press the “OK” button.

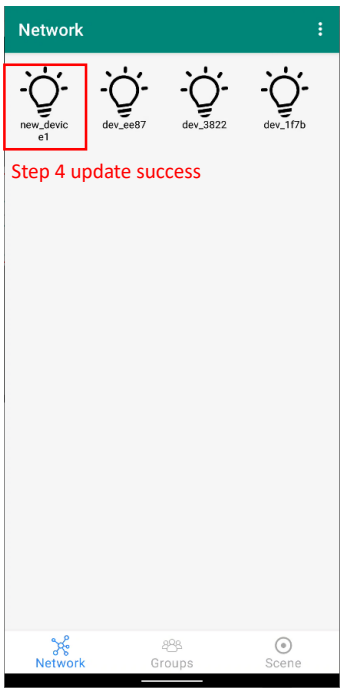
➢ Step 4

Go back to the Network page and check whether the update is successful.









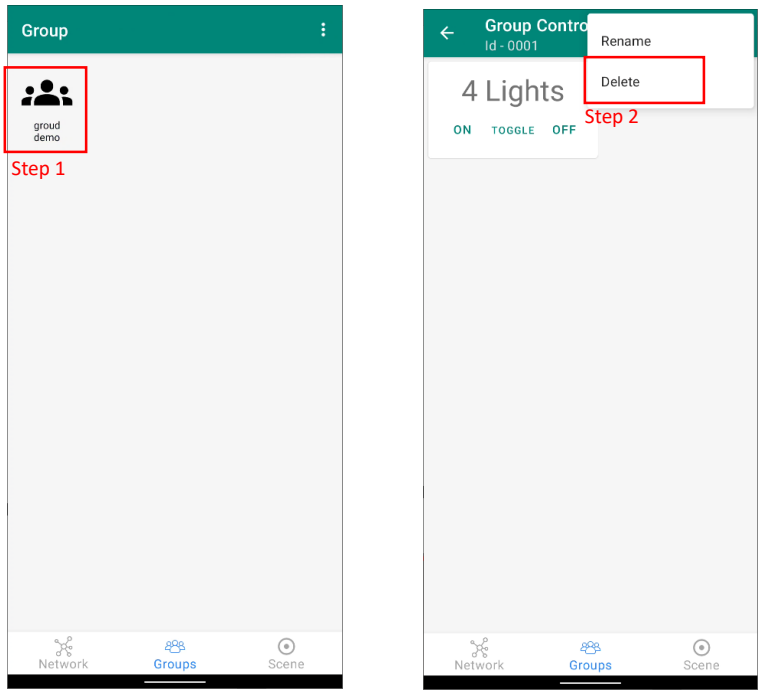
- Remove group

➢ Step 1

Select the group you want to remove, press the icon and enter the operation page.

➢ Step 2

Press the “Delete” button on the Menu.



- Rename group

➢ Step 1

Select the group you want to rename, press the icon and enter the operation page.

➢ Step 2

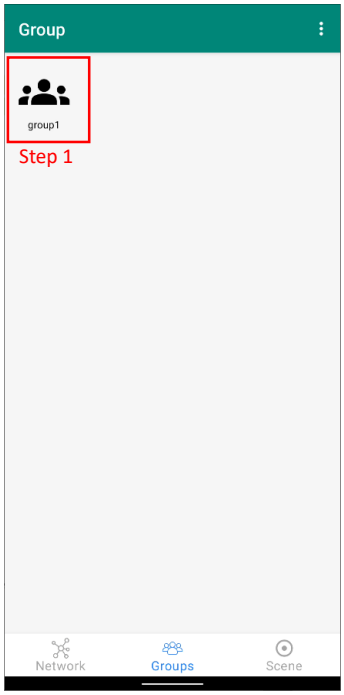
Press the “Rename” button on the Menu.

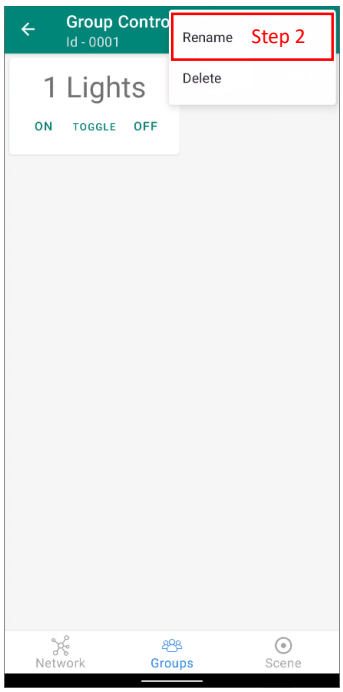
➢ Step 3

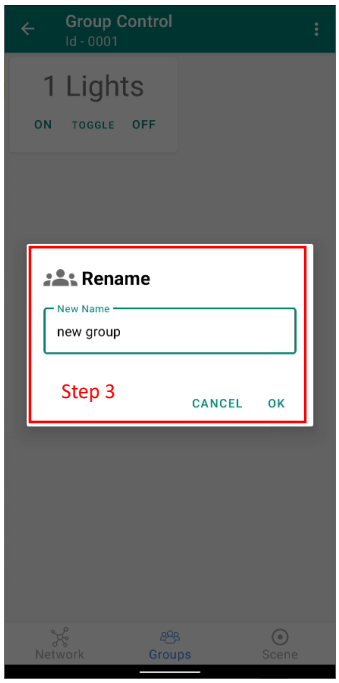
Set a new name and press the “OK” button.

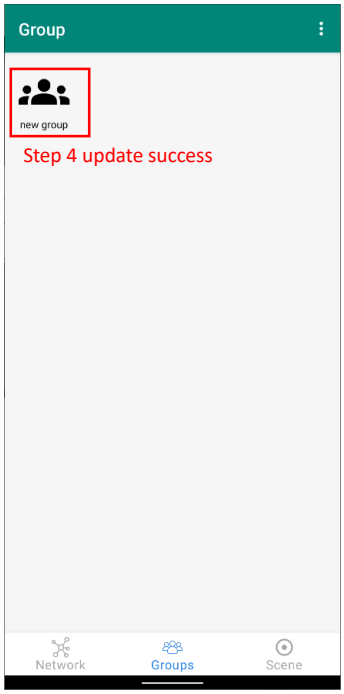
➢ Step 4

Go back to the “Groups” page and check whether the update is successful.









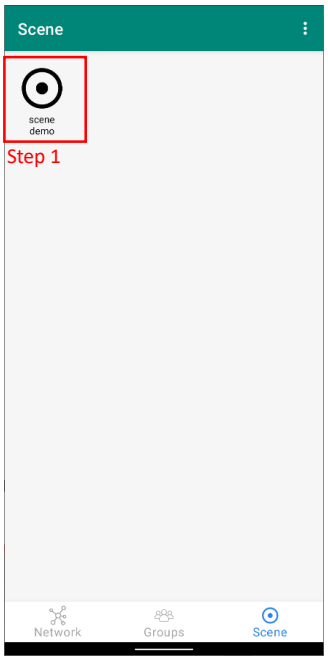
- Remove scene

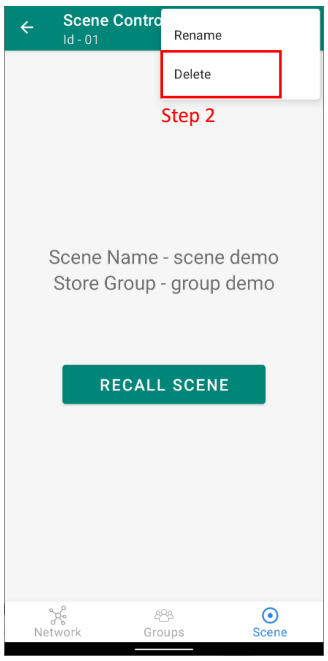
➢ Step 1

Select the scene you want to remove, press the icon and enter the operation page.

➢ Step 2

Press the “Delete” button on the Menu.





- Rename scene

➢ Step 1

Select the scene you want to rename, press the icon and enter the operation page.

➢ Step 2

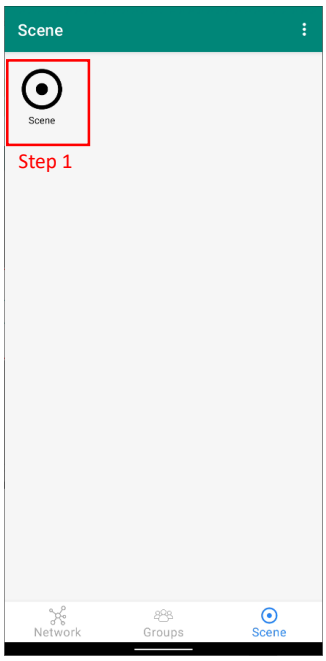
Press the “Rename” button on the Menu.

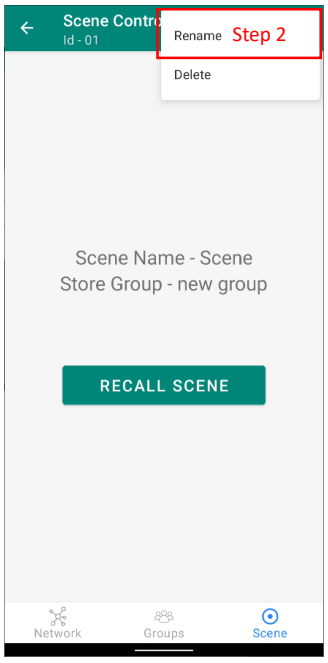
➢ Step 3

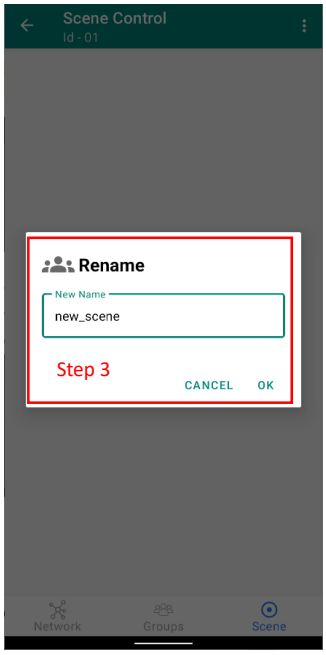
Set a new name and press the “OK” button.

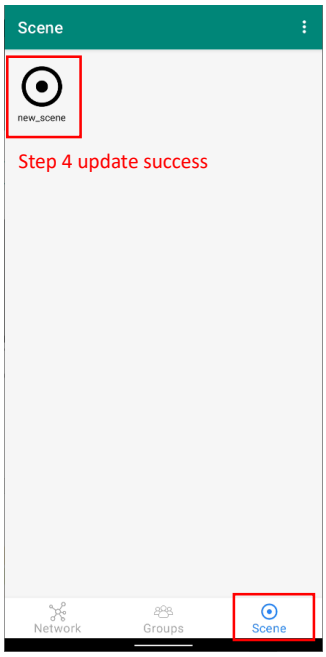
➢ Step 4

Go back to the “Scene” page and check whether the update is successful.









- Reset gateway

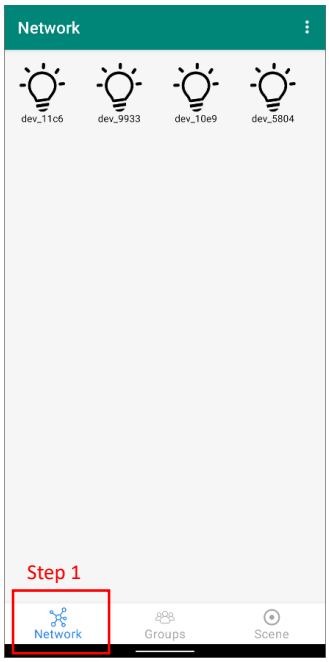
  Gateway reset will clear all device information, as well as group and scene information.

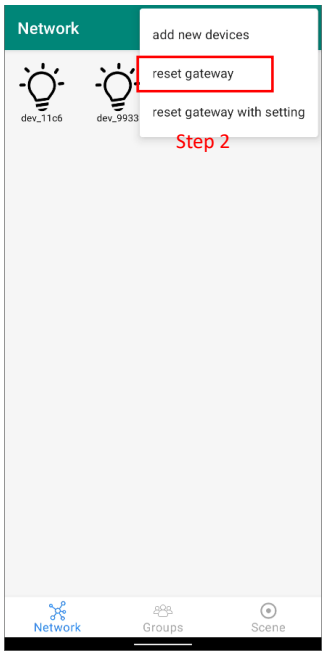
➢ Step 1

Press the "Network" button.

➢ Step 2

Press the “reset gateway” button on the Menu.





- Reset gateway with setting

The "Reset gateway with setting" function includes clear all device information and reassigning the communication channel and pan id.

- Step 1

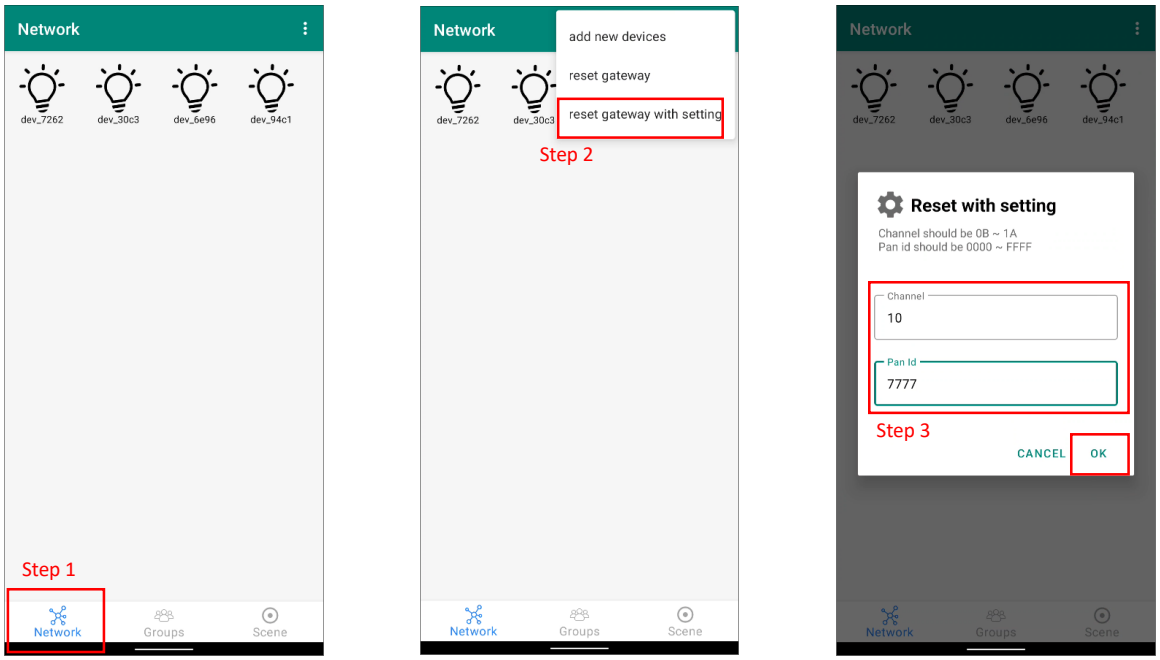
  Press the "Network" button.

- Step 2

  Press the “reset gateway” button on the Menu.

- Step 3

  Enter Channel (default is 0x0C), pan ID and press OK.

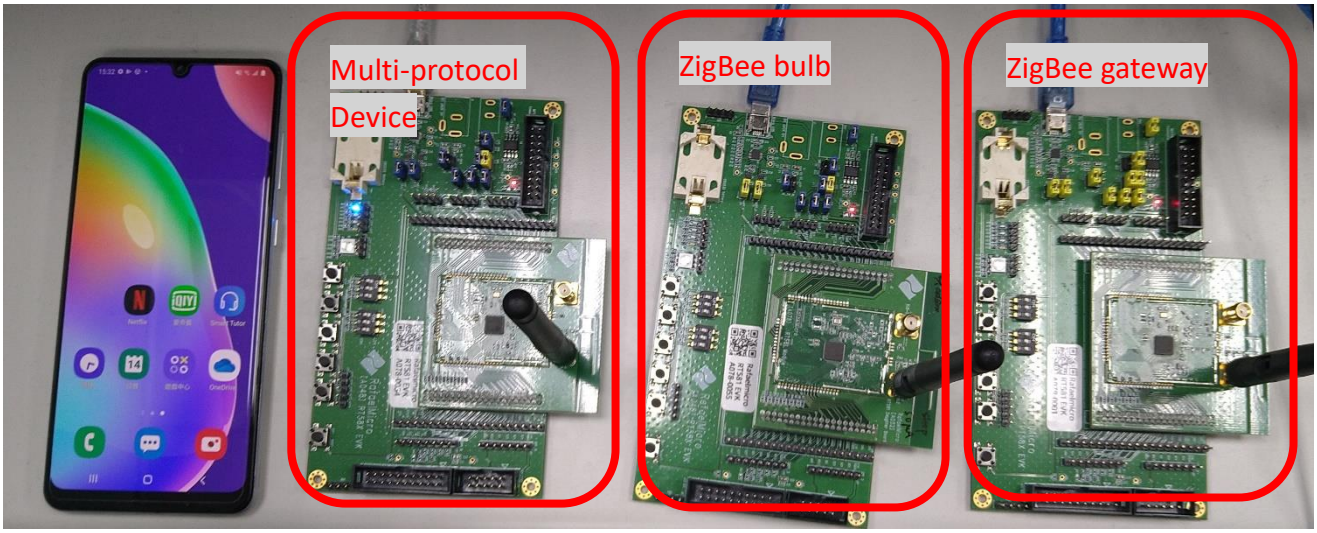


Users can observe from the device terminal log whether the operation is successful.

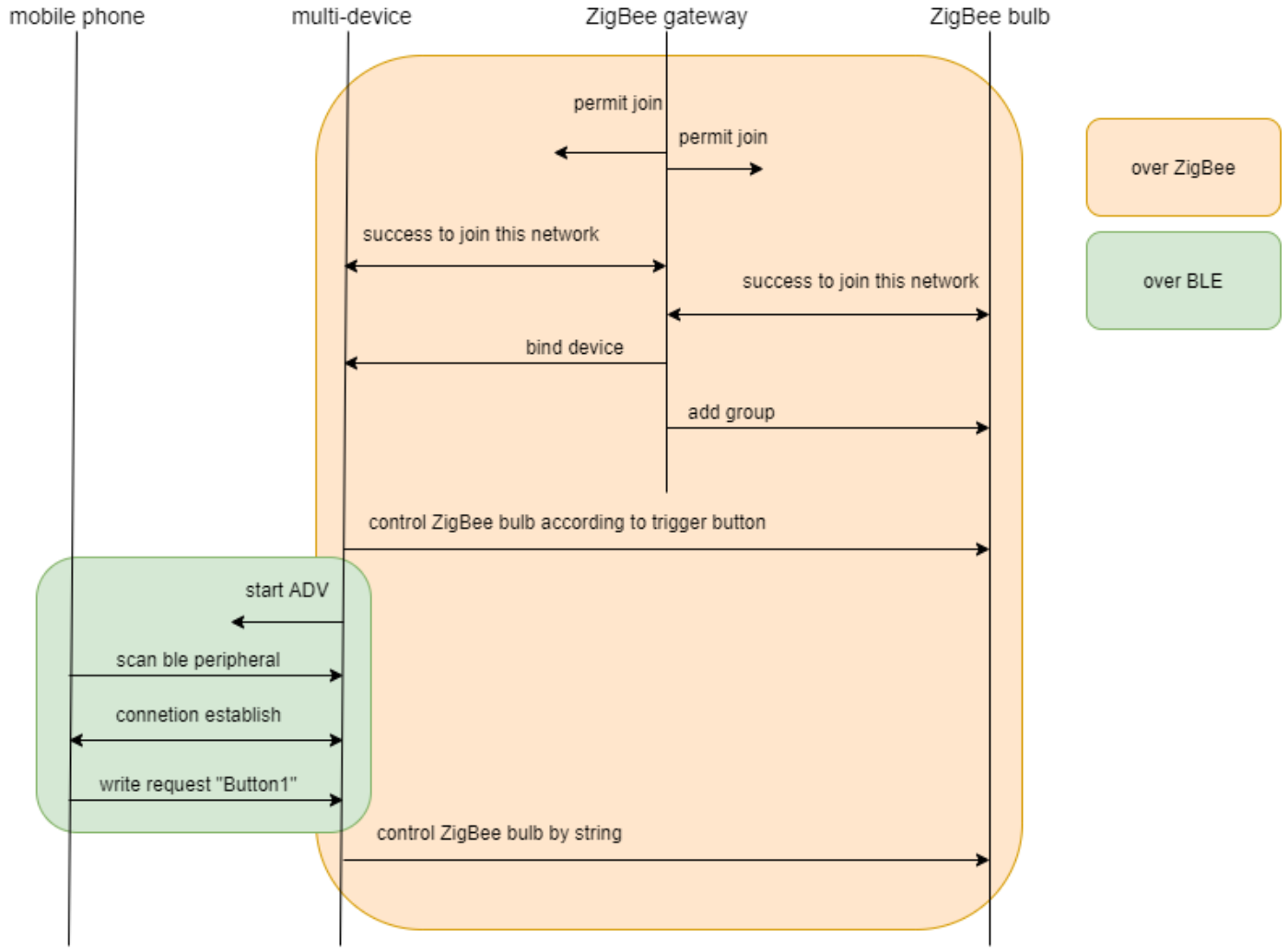
```
_cmd_zc_info_handle 2
PAN ID 0x7777
Channel 144
zigbee_nwk_start_request( PAN: 7777, ch:16, reset:1 65536 0)
Z< scan channel page 0, channels 0x00010000, scan_duration 5
Advertising...
permit_join duration = 0s
permit_join duration = 180s
Device start success
PAN: 7777, ShortAddr: 0000, on channel: 16, MAC: A0:44:F3:FF:FF:FC:00:CB
------------------------ >>>> GW      ------------------------

  2000B84C | FF FC FC FF 07 04 0A 00 - 10 00 00 00

_cmd_zc_info_handle 4
------------------------ >>>> GW      ------------------------

  2000B864 | FF FC FC FF 0B 03 0A 00 - 10 00 00 00 77 77 10 00

_cmd_zc_info_handle 3
permit_join duration = 0s
```

4.8 Running Demo ZED_Switch_BLE_TRSP

This demo will use at least three EVK board and one smart phone, these three EVK play the roles of ZigBee gateway, ZigBee light and multi-protocol device (ZigBee switch & BLE Transparent) respectively.



We can control the ZigBee light via the buttons of the multi-protocol device or using smart phone via BLE.



4.8.1 Form a ZigBee network

Complete the following steps to connect “ZigBee gateway”:

- Connect the RT58x EVK boards to the USB ports.
- Start the serial tools (Putty, Tera Term)
- Configure the baud rate to 115200, none, 1.

Send command to gateway to form a ZigBee network

Command format:

start [Erase NWK info] [Channel] [PAN ID] [Max child]

Parameter:

[Erase NWK info]:

1 => erase the legacy network information,

0 => reuse legacy network information.

[Channel]: set different channel from 11~26.

[PAN ID]: 16-bits PAN ID of zigbee network.

[Max child]: maximum supported child number.

Example: start 1 12 0x1213 30

4.8.2 Join ZigBee switch and Light into ZigBee network

Send command to gateway to permit device join in specific time

Command format:

pj [Enable] [Timeout]

Parameter:

[Enable]:

1 => allow device join,

0 => not allow device join.

[Timeout]: allow device join time (second).

Example: pj 1 60

after device successful join, the console will show following information to let user know which device is light or switch. Device ID 0x0102 is for ZigBee light and ZigBee switch using Device ID 0x0104/0x0105.

```
>>zdo_signal_handler: status 0 signal 48
Device Announce :
        IEEE 12:74:45:32:33:32:38:50
        Short address 0x05c5, Cap 80
Active Ep : Addr 5C5, Endpoint 010203
Simple desc : Addr 5C5, Endpoint 01, Profile 0104, DeviceID 0105
Simple desc : Addr 5C5, Endpoint 02, Profile 0104, DeviceID 0104
Simple desc : Addr 5C5, Endpoint 03, Profile 0104, DeviceID 0105
>>zdo_signal_handler: status 0 signal 47
>>zdo_signal_handler: status 0 signal 18
>>zdo_signal_handler: status 0 signal 48
Device Announce :
        IEEE 11:35:45:32:33:32:38:50
        Short address 0x4e8d, Cap 8E
Active Ep : Addr 4E8D, Endpoint 02
Simple desc : Addr 4E8D, Endpoint 02, Profile 0104, DeviceID 0102
>>zdo_signal_handler: status 0 signal 47
group a 0x4e8d 2 0x0001
Recv ZCL message 0x0000
```

4.8.3 Add ZigBee light into specific group

Command format:

group [action] [target address] [target endpoint] [group id]

Parameter:

[action]: a => add, r => remove.

[target]: address: 16bits network address.
[target]: endpoint: 8bits endpoint.
[group id]: 16bits group ID.
Example: group a 0x4e8d 2 0001

4.8.4 Bind the cluster of ZigBee switch to specific group

Command format:

bind [target address mode] [src address] [src endpoint] [target address] [target endpoint] [cluster id]

Parameter:

[target address mode]: Destination address mode (0x01/0x03 =>16bits group/network address).
[src address]: 16bits source network address.
[src endpoint]: 8bits source endpoint.
[target address]: 16bits group/network address.
[target endpoint]: 8bits destination endpoint (any value when destination address mode is 0x01)
[cluster id]: 16bits cluster ID.

Example:

1. Switch 0x05c5 endpoint 1 bind on/off cluster to group address 0x0001:

bind 0x01 0x05c5 1 0x0001 0x00 0x0006

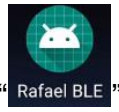
2. Switch 0x05c5 endpoint 2 bind level cluster to group address 0x0001:

bind 0x01 0x05c5 2 0x0001 0x00 0x0008

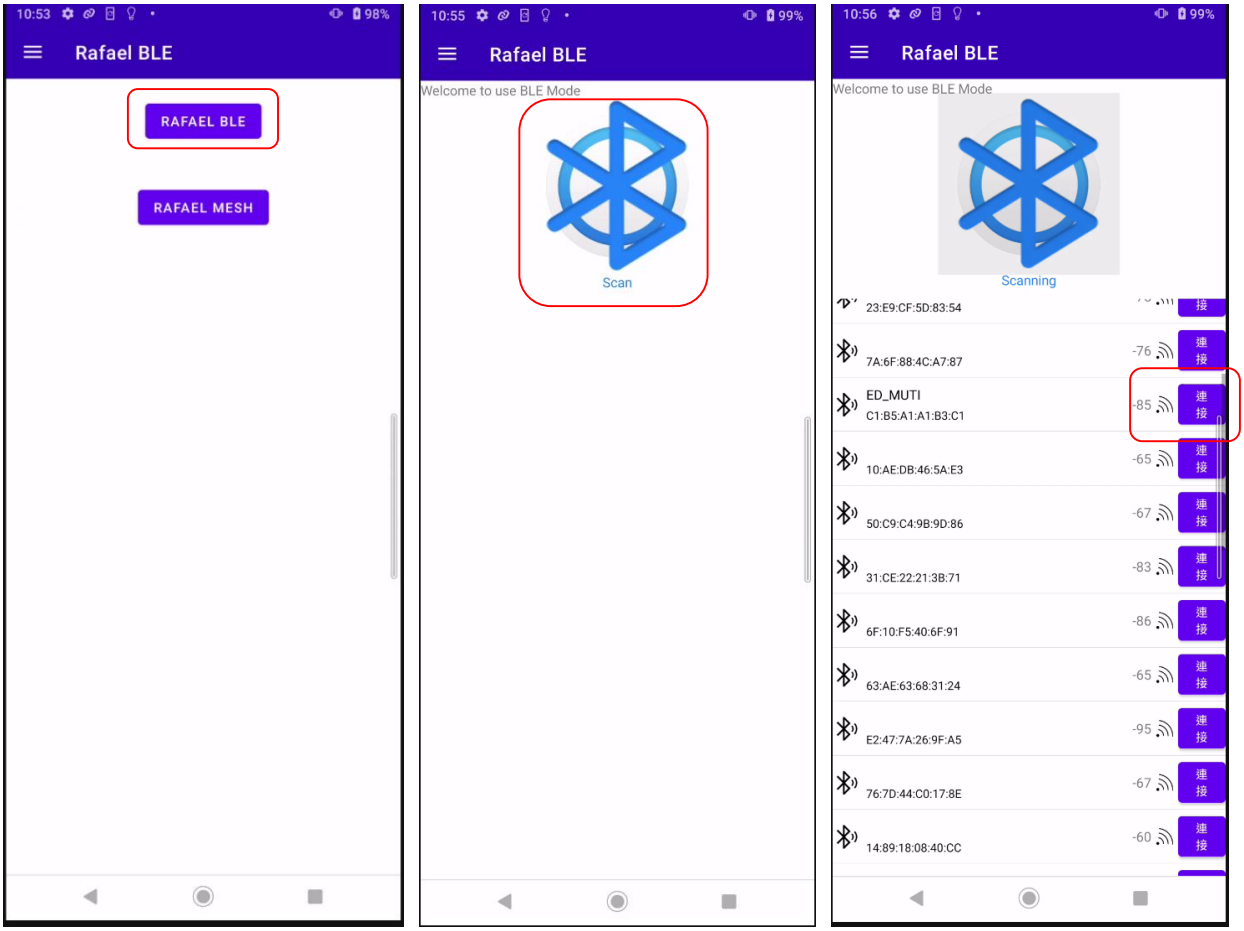
3. Switch 0x05c5 endpoint 3 bind color temperature cluster to network address 0x1233 endpoint 2:

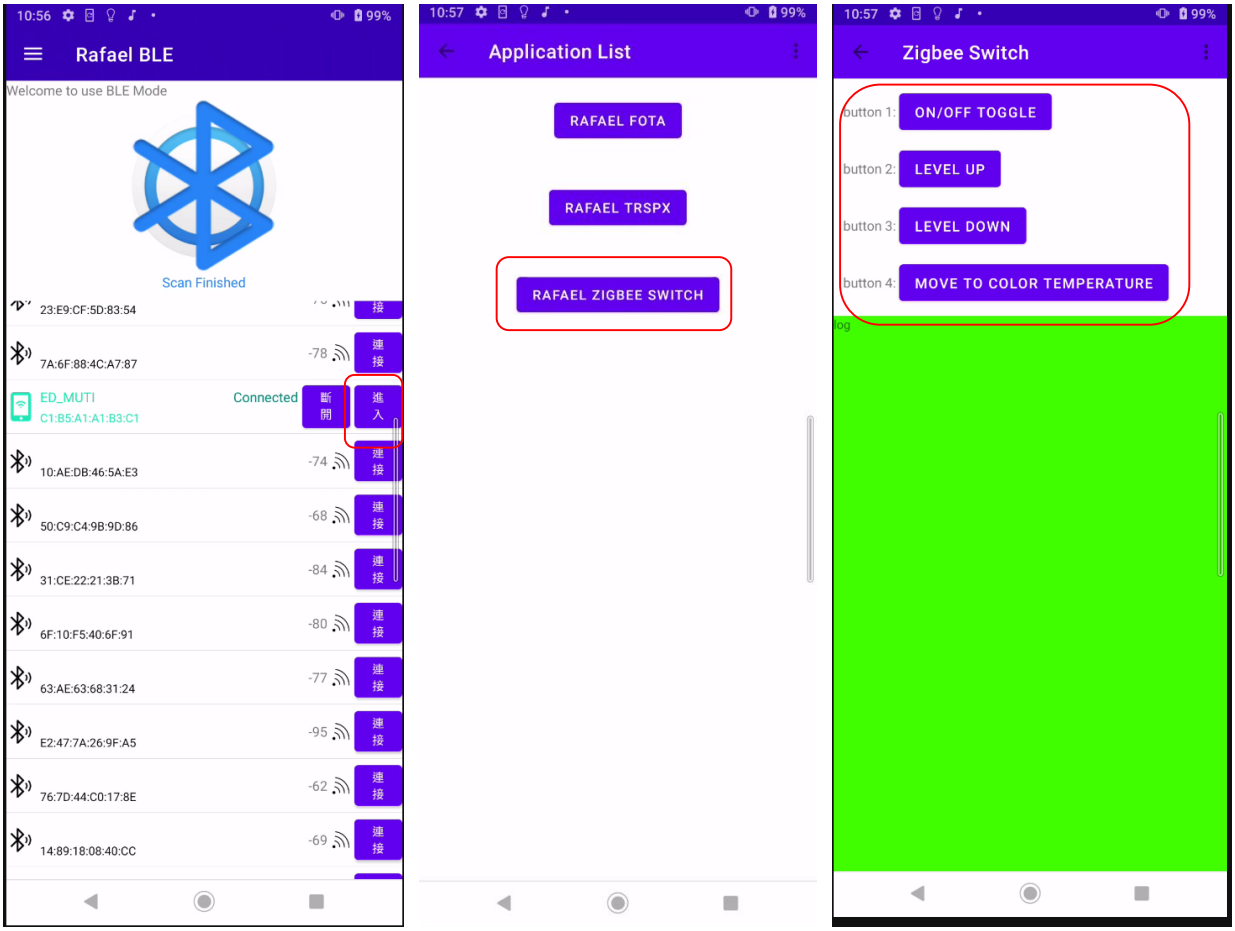
bind 0x03 0x05c5 3 0x1233 0x02 0x0300

4.8.5 Control ZigBee light over BLE

After joining multi-protocol device to ZigBee network, you can use mobile app “ 

” to connect the BLE device “ED_MULTI”





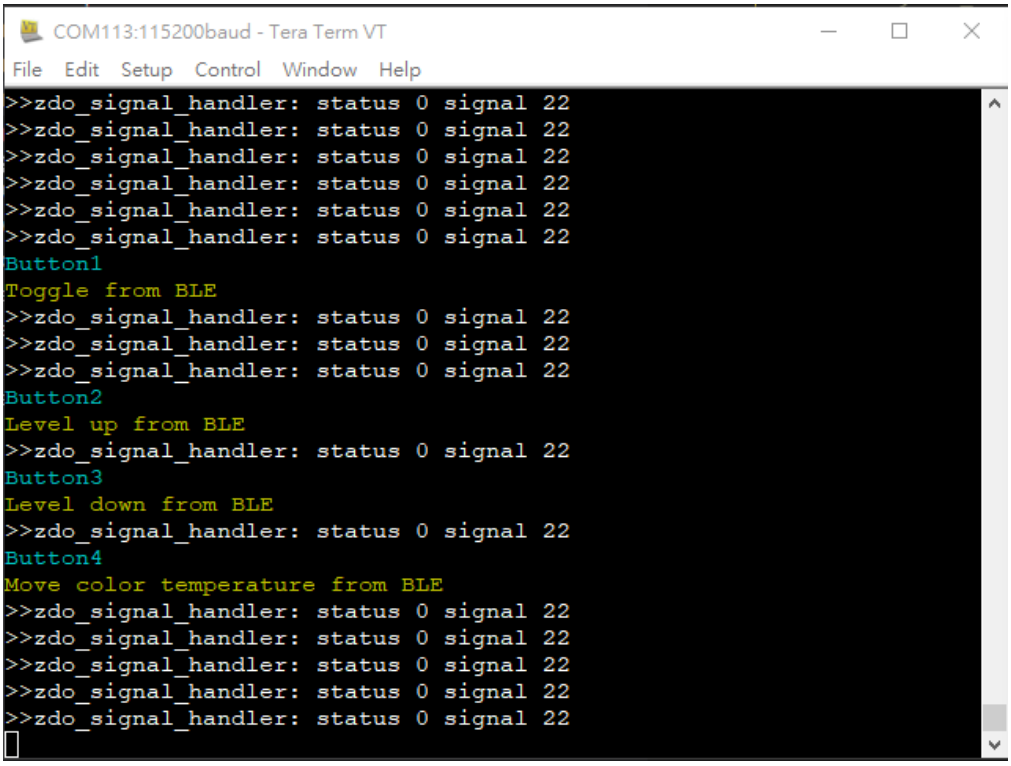
Mobile APP show 4 icons on/off toggle, level up, level down and move to color temperature which corresponding to button of EVK board. And these 4 icons were representative to 4 different data string “Button1”, “Button2”, “Button3” and “Button4”. We can touch each icon to send these data string to multi-protocol device over BLE.

In ble_app.c, we also define these data string “Button1” ~” Button4” to emulate the button of switch triggered.

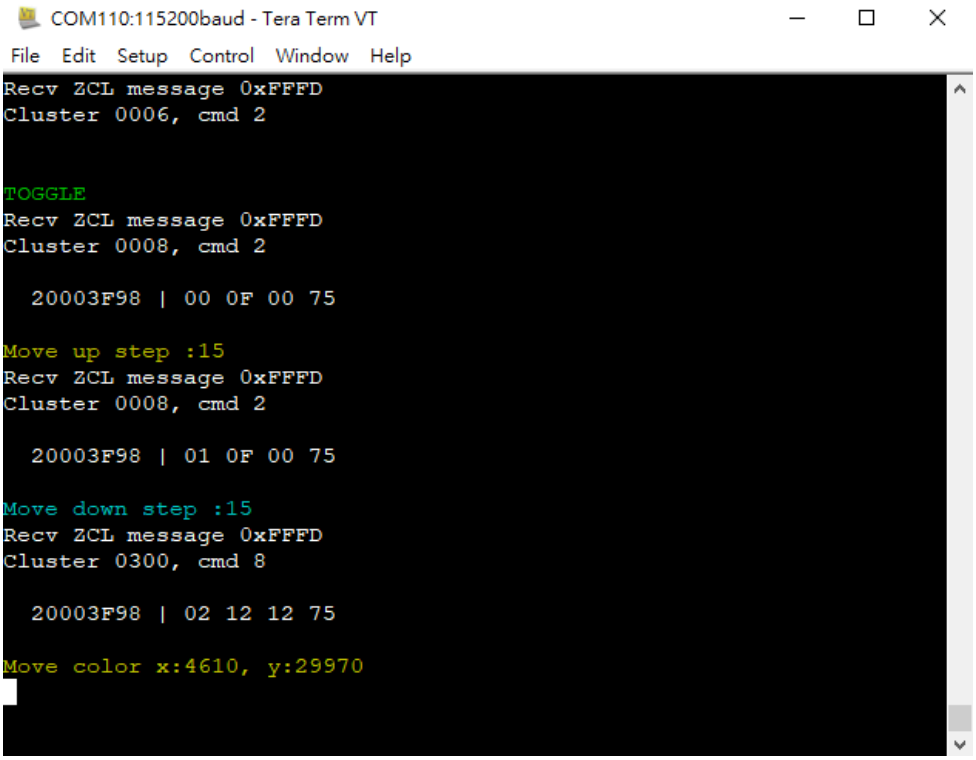
```
#define BUTTON_1_STR            "Button1"
#define BUTTON_2_STR            "Button2"
#define BUTTON_3_STR            "Button3"
#define BUTTON_4_STR            "Button4"
```

So that when device received these string it will convert to ZigBee command and send to bulb over ZigBee

Console of multi-protocol device:



Console of ZigBee light:



4.9 Running Demo ZED_Switch_BLE_TRSP_FOTA

The detail step of FOTA update procedure is same as BLE demo “TRSP_FOTA_Periph”. So that you can reference document “**[SW_08] RT58x_BLE_SDK_OTA_Guide_V1.0**” to complete the FOTA update procedure over BLE. Except for one limitation which is FOTA update procedure cannot perform when ZigBee protocol is trying to join/rejoin ZigBee network.

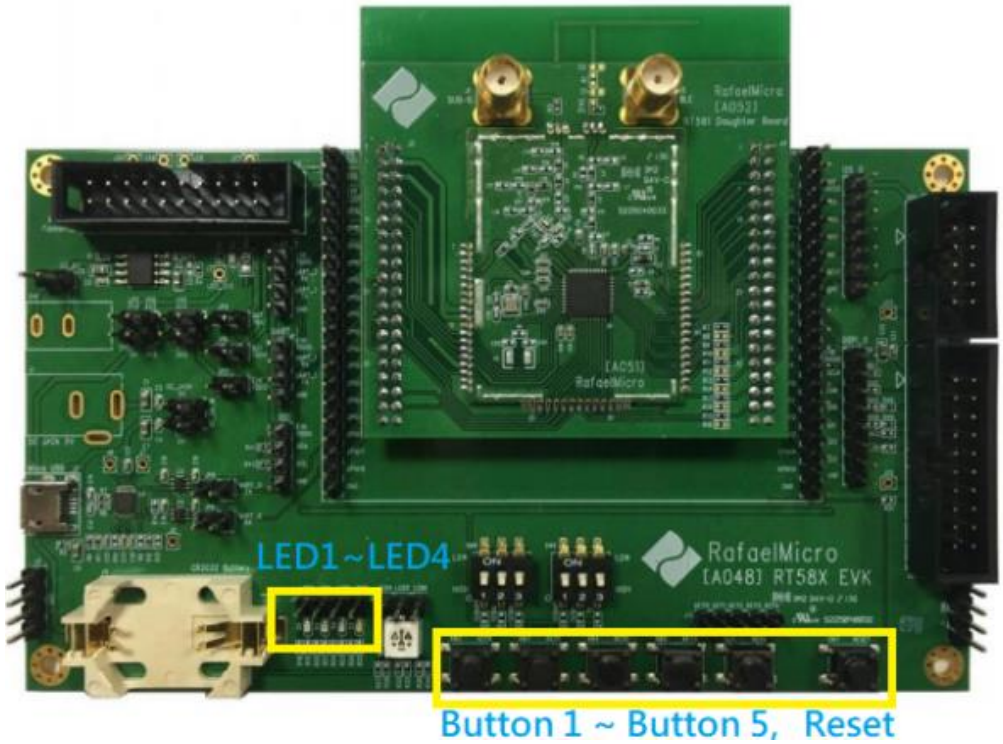
If multi-protocol device was not join any ZigBee network, after power on it will try to search ZigBee network for maximum 3 times, if there were not suitable network can be join, it will stop this procedure.

```
------------------------------------------------
  ZigBee Switch + BLE TRSP + FOTA (P) demo: start...
------------------------------------------------
Initial ZigBee/BLE stack
Create app task
timer_low_running_handler(0)
sysinfo: sys ver 0001
Write default data length, status: 0
Advertising...
Device do join
Device do join
Device do join
Device suspend join
```

If multi-protocol device already joined into ZigBee network, after power on it will try to rejoin original network for maximum 5 times, if no suitable parent found, it will stop this procedure.

```
------------------------------------------------
  ZigBee Switch + BLE TRSP + FOTA (P) demo: start...
------------------------------------------------
Initial ZigBee/BLE stack
Create app task
timer_low_running_handler(0)
fw buffer not empty!
sysinfo: sys ver 0001
Write default data length, status: 0
Advertising...
Device do rejoin
Device do rejoin
Device do rejoin
Device do rejoin
Device do rejoin
Device suspend rejoin
```

The join/rejoin procedure can be restart by the button 5 of EVK



4.10 Running Demo ZR_Light_BLE_TRSP

The detail joining multi-protocol device procedure is same as ZigBee demo “Light”. You can reference document “[SW_15]RT58x_Zigbee_Gateway_Light_Switch_Operation_Guide_V1.0” or follow the step of this document section 4.8.2 to join multi-protocol device into ZigBee network

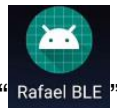
This demo use at least two EVK board and one smart phone, these two EVK play the roles of ZigBee gateway, multi-protocol device (ZigBee light & BLE Transparent).

4.10.1 Multi-device BLE ADV enable

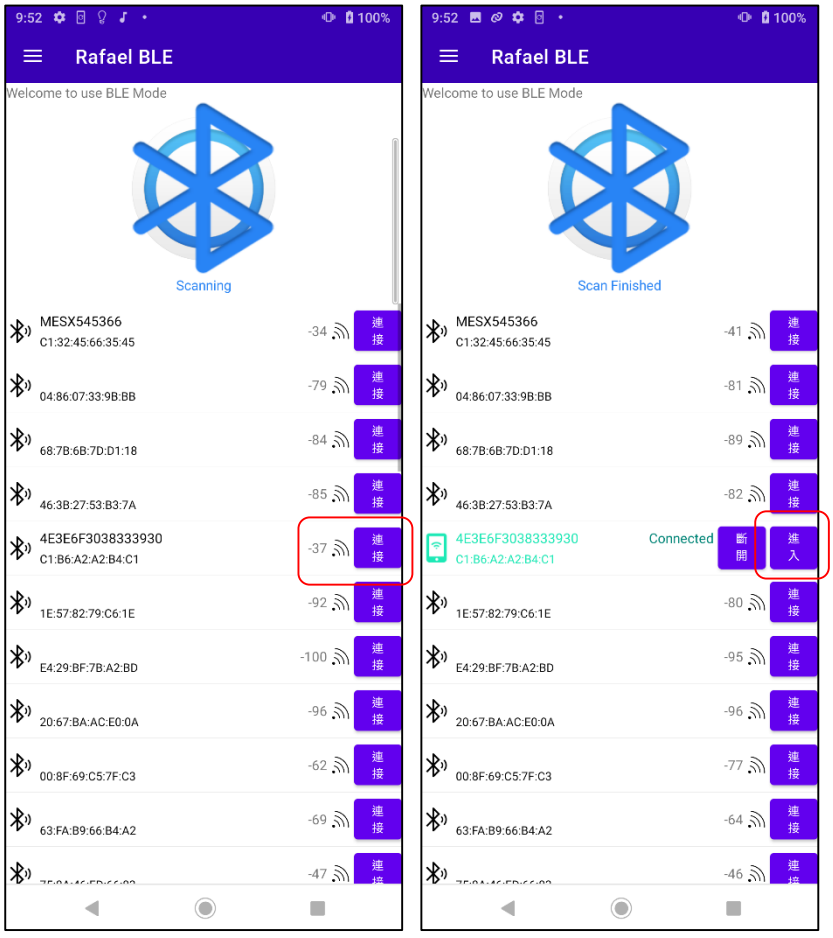
Multi-protocol device was not always enable BLE ADV. BLE ADV is only enabled for the first minute after power up if the multi-protocol device has joined the ZigBee network.
And the BLE scan response is set according to the ZigBee MAC address, which is convenient for users to find the device that needs to be connected.

```
------------------------------------------
  ZigBee light + BLE TRSP (P) demo: start...
------------------------------------------
Initial ZigBee/BLE stack
Create app task
File System position : F0000, size : 4096
Magic correct
timer_low_running_handler(0)
Write default data length, status: 0
Device join success
PAN: 1234, ShortAddr: 6106, on channel: 11, MAC: 4E:3E:6F:30:38:33:39:30
mac addr 30:39:33:38:30:6F:3E:4E
Advertising...
```

4.10.2 Reset ZigBee information to factory new over BLE

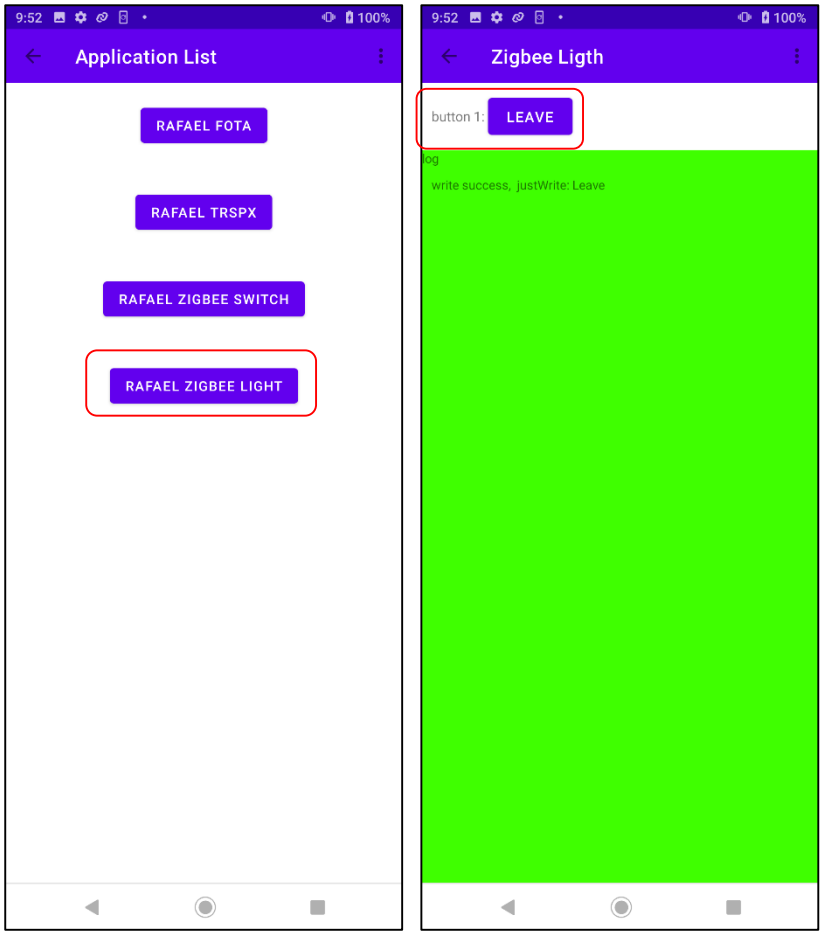
Using mobile app “ 

” to connect the BLE device



After connected over BLE, go to page “RAFAEL ZIGBEE LIGHT” and use button” LEAVE” to

force multi-protocol device reset ZigBee information to factory new



After multi-protocol device reset to factory new, it will perform system reboot, and BLE connection also terminated.

```
Connected, ID=0, Connected to 5f:5a:d7:58:2e:b4
Data length changed, ID: 0
MaxTxOctets: 251  MaxTxTime:2120
MaxRxOctets: 27  MaxRxTime:328
Connection updated
ID: 0, Interval: 6, Latency: 0, Supervision Timeout: 500
Connection updated
ID: 0, Interval: 36, Latency: 0, Supervision Timeout: 500
adv already stop
Leave
leave from BLE
Device Leave :
        IEEE 4E:3E:6F:30:38:33:39:30
        Short address 0xffff, rejoin 00
clear scene D? ------------------------------------------
  ZigBee light + BLE TRSP (P) demo: start...
------------------------------------------
```

Revision History

Revision	Description	Owner	Date
1.0	Initial version	Nat	2022/02/11
1.1	Add command format of ZigBee gateway	Nat	2022/03/14
1.2	Add new demonstration	Nat	2022/07/19
1.3	Add smart home app user interface	Ryan	2022/12/01
1.4	Add more explanations for the first time connecting to GW	Nat	2023/02/13