臺北科技大學資訊工程系

110 年度資工系學生受疫情影響無法進行校外實習
之無線及寬頻網路實驗室實務專案報告

異質物聯網系統的評估與實現

專題編號：—

專題計劃參與人員：107590057　李振豪

107590035　許哲維

指導教授：吳和庭 教授

執行期間：110 年 7 月 12 日至 110 年 9 月 3 日

章節目錄

異質物聯網系統的評估與實現

專題編號：—
執行期限：110 年 7 月 12 日至 110 年 9 月 3 日
指導教授：吳和庭 教授
專題參與人員： 107590057 李振豪
107590035 許哲維

一、背景

這些年，無線相關產品蓬勃的發展，就連家庭的設備也趨近於無線化的趨勢。無線感測網路(Wireless Sensor Network)是一種結合感測、運算及網路能力的技術，由一到多數個無線資料接收器所構成的網路，將每個感測器串聯成感測網路，利用感測器的感應範圍，偵測週遭環境及特定目標，將一些的簡易資料做運算的動作之後，回傳到資料接收器，進而讓我們方便擷取資料，並進一步透過所獲得資料來瞭解環境的狀況，透過這些資料來做調整、維護或提供更好的服務。為了達到接收器大量的佈置，必須具備低成本、低功耗、體積小、容易佈建，並具有感測環境裝置、可程式化控制、可動態組成的特性。

二、研究動機和目的

異質物聯網系統是近年來熱門的應用領域，然而該如何用一個軟硬體平台進行整合及接收 ZigBee(紫蜂)與 BLE(藍牙低功耗)網路所傳的訊號是本專案所著墨的重點。

因此，我們以 Zephyr OS 為專案基礎，並透過 nRF52840 DK 開發套件來實現系統與訊號發送器，以便我們評估本專案之可行性與其未來展望。

在滿足工業物聯網(IoT)的無線感測環境下，實現利用一個軟硬體平台—異質物聯網系統，一個可整合及接收 ZigBee 及藍牙低功耗網路所傳輸的訊號，並以此評估異質物聯網系統之可行性與未來展望。

關鍵詞： 無線感測網路、異質物聯網系統、藍牙低功耗、ZigBee、Zephyr OS、nRF52840 DK。

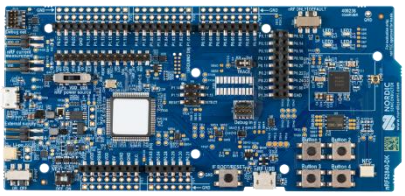
三、實驗開發工具與環境

(一)、Zephyr OS

Zephyr 是一個小型的即時作業系統，用於資源受限的嵌入式互聯裝置，支援多種體系並在 Apache 許可證 2.0 下發行。
https://www.zephyrproject.org/

(二)、nRF52840 DK

nRF52840 DK 為一個可應用藍牙低功耗和 ZigBee 之無線控制晶片開發套件。



(三)、nRF Connect SDK

nRF Connect SDK 是一個可擴展的統一軟體開發套件，用於構建基於 nRF52、nRF53 和 nRF91 系列等無線設備的產品。
https://tinyurl.com/723yah23

(四)、CMake

是個一個開源的跨平台自動化建構系統，用來管理軟體

建置的程式，並不依賴於某特定編譯器，並可支援多層目錄、多個應用程式與多個函式庫。

本次專案使用的版本為：3.20.0

(五)、**Python**

是一種廣泛使用的直譯式、進階和通用的程式語言。Python 支援多種程式設計範式，包括函數式、指令式、結構化、物件導向和反射式程式。

直譯器本身幾乎可以在所有的作業系統中執行。

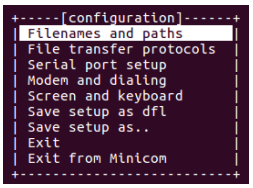
本次專案使用的版本為：3.8.10

(六)、**Devicetree compiler**

將適合人類閱讀和編輯的 DTS 檔案編譯成適合機器處理的 DTB 檔案。

本次專案使用的版本為：1.5.0

(七)、**Minicom**



可用來與串口設備通信，如調試交換機和 Modem 等。**可察看目前硬體設備的狀態和信息。**

四、相關技術

(一)、**C**

是一種通用的程式語言，廣泛用於系統軟體與應用軟體的開發。本專案之程式皆以 C 作為基礎。

(二)、**藍牙低功耗(BLE)**

是一種個人區域網路技術，旨在用於醫療保健、運動健身、信標、安防、家庭娛樂等領域的新興應用。相較一般藍牙，低功耗藍牙旨在保持同等通訊範圍的同時顯著降低功耗和成本。

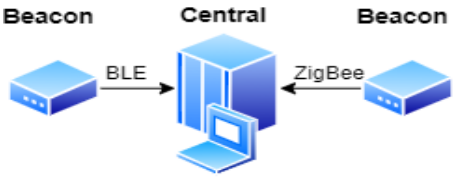
(三)、**ZigBee**

是一種低速短距離傳輸的無線網路協定，底層是採用 IEEE 802.15.4，低速、低耗電、低成本為其特色。其傳輸距離約為數十公尺,使用頻段為免費的 2.4GHz 與 900MHz 頻段，傳輸速率為 10K 至 250Kbps，網路架構具備 Master/Slave 屬性，達到雙向通信功用。

(四)、ZBOSS Open Initiative

是一款便攜式，高性能 Zigbee 軟件協議棧，具有跨平台支持，多任務處理，固定內存占用，無操作系統配置以及易於使用的應用程序接口(API)。

五、架構流程



(一)、藍牙低功耗訊號發送

傳送 BLE 之訊號。

(二)、訊號接收系統

接收 BLE 或 ZigBee 所傳之訊號。

(三)、ZigBee 訊號發送

傳送 ZigBee 之訊號。

六、專案目標

於一個 Zephyr OS 平台上同時收到 BLE 及 Zigbee 的 RSSI 訊號，並整合在 Central 裝置的頁面中呈現出來。

七、實驗步驟與流程

- 開啟 command line,依序安裝 Zephyr 所需軟體與套件，並設定相關之環境變數。

1. 更新 ubuntu

sudo apt update

sudo apt upgrade

2. 安裝工具包

sudo apt install --no-install-recommends git cmake ninja-build gperf \

ccache dfu-util device-tree-compiler wget \

python3-dev python3-pip python3-setuptools python3-tk python3-wheel xz-utils file \

make gcc gcc-multilib g++-multilib libsdl2-dev

3. 安裝 west 工具

pip3 install --user west

4. 建立一個 ncs 的資料夾

cd ~

mkdir ncs

5. 下載 zephyrproject 源碼

cd ncs

west init -m https://github.com/nrfconnect/sdk-nrf --mr NCS_revision

west update

6. 導出 cmake 包

west zephyr-export

7. 使用 pip3 安裝其他依賴的工具包

pip3 install --user -r

zephyr/scripts/requirements.txt

pip3 install --user -r nrf/scripts/requirements.txt

pip3 install --user -r

bootloader/mcuboot/scripts/requirements.txt

8. 添加變量

export

ZEPHYR_TOOLCHAIN_VARIANT=gnuarm

emb

export

GNUARMEMB_TOOLCHAIN_PATH="~/gn

uarmemb/version-folder"

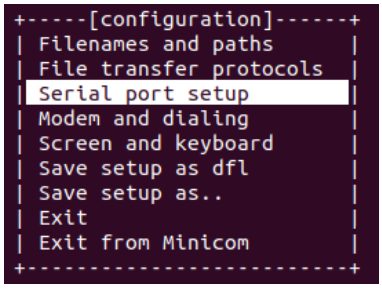
- Minicom 安裝與設定

1. 安裝 Minicom

sudo apt-get install minicom

2. 設定 Minicom

sudo minicom -s



進入 Serial port setup

按 A 更改裝置的位置

例如 使用 USB 故所抓到的是 ttyUSB0

在這裡抓到的是 ttyACM1

按 E 更改成我們所需的 Bps/Par/Bits 這邊是 115200 8N1

按 F 將 Hardware flow control 設為否

完成後要記得儲存設定值 save setup as dfl。

- 將欲燒錄之 nRF52840 DK 以傳輸線連結至已安裝 Zephyr 相關套件之電腦。之後便可以打開相對應 USB 編號的 Minicom 檢視。

開啟 minicom

sudo minicom

如有第二個硬體，另外開啟新的 command line 輸入：

(名稱為 ttyACM2)

sudo minicom -D /dev/ttyACM2

以此類推，第三個硬體，輸入：

sudo minicom -D /dev/ttyACM3

- 透過 Ubuntu 之終端機打開資料夾並依序編譯程式碼(Build)。

1. 進入目標資料夾

cd ~/ncs/zephyr

2. 將本次專案所需的檔案逐項拉進 ~/ncs/zephyr/samples

3. 燒錄 light_switch 至 nRF52840 DK 作

為 ZigBee 訊號發送端。

west build -b nrf52840dk_nrf52840 samples/light_switch/ --pristine

west flash

4. **燒錄 ble_beacon 至 nRF52840 DK 作為藍牙低功耗訊號發送端。**

west build -b nrf52840dk_nrf52840 samples/ble_beacon/ --pristine

west flash

5. **燒錄 testctrN 至 nRF52840 DK 作為訊號接收端。**

west build -b nrf52840dk_nrf52840 samples/testctrN/ --pristine

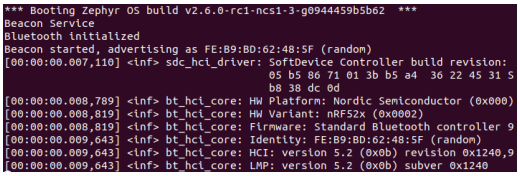
west flash

6. **打開燒錄 testctl 之 nRF52840 DK 連結的 Ubuntu minicom 觀察接收情況，其結果如下面附圖。**

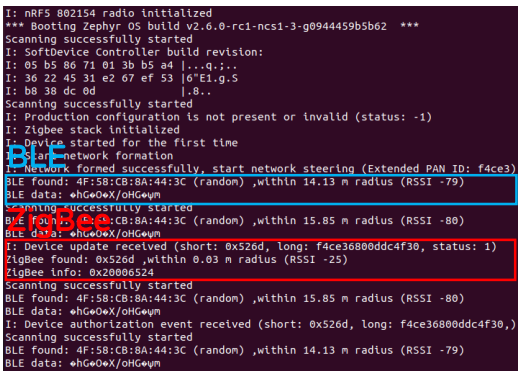
sudo minicom

八、實驗結果與效能

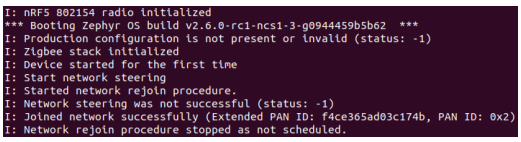
- 藍牙低功耗訊號發送(Ble_beacon)



- 訊號接收系統(Central)



- ZigBee 訊號發送(Light_switch)



九、問題與解決

- 團隊溝通問題

在剛開始時，因為還在摸索 Zephyr 的操作，所以兩人都在研究相同的東西，並沒有明確分工，導致進度較為緩慢。

Sol. 討論並依項目分工。

- ZigBee 資源不足

由於不清楚 Zephyr 之 ZigBee 傳輸之程式架構，但其提供之 Sample 苦無相關範例難以從中學習。

Sol. 尋找其他範例。

- 系統整合問替

整合時，同時部分產生一些問題。

Sol. 調整部分程式碼。

十、結論

經過結果可知其異質物聯網系統是可行的，但在專案進行中，發現 Zephyr OS 在 ZigBee 資源相較於藍牙低功耗是較為貧乏的，因此也提高了我們在實作上的難度，但藉由一些例子讓我們能夠實現本專案之功能，因此，本系統評估結果是可行的。

十一、 未來展望

- 系統端程式實用化
- 訊號發送端系統之程式設計

十二、 參考文獻

[1] Zephyr, “Zephyr Project” , Internet: https://tinyurl.com/27b448u5.

[2] WikipediA, “Zephyr” , Internet: https://zh.wikipedia.org/wiki/Zephyr, 2016-2021.

[3] NORDIC, “nRF52840 DK”, Internet: https://tinyurl.com/wv9hszcd.

[4] ASK UBUNTU, Internet:

https://askubuntu.com/.

[5] Coldnew's blog,「Zephyr RTOS 開發記錄：基本環境建立」, Internet: https://coldnew.github.io/7f7c0c84/, 2016.

[6] Zephyr Project API Documentation, Internet: https://tinyurl.com/8ya37aj6.

[7] nRF Connect SDK documentation, Internet: https://tinyurl.com/435vddbu.

[8] Developing with ZBOSS, Internet: https://tinyurl.com/bbr9wmss.

[9] nRF Connect SDK, Internet: https://tinyurl.com/723yah23.

[10] Bluetooth Low Energy – Wikipedia, Internet: https://en.wikipedia.org/wiki/Bluetooth_Low_Energy.

[11] ZBOSS Open Initiative, Internet: https://dsr-zoi.com/

附錄

ble_beacon

```
/* main.c - Application main entry point */

/*
 * ReadMe_Ble_beacon
 * |Overview
 *  測試 BLE 之 Beacon 的角色
 * |Building and Running
 * cd Ble_beacon
 * west build -b nrf52840dk_nrf52840
 * west flash
 * sudo minicom -D /dev/<your device>
 *  其發送狀況可於 Ubuntu 終端機呈現
 */

/*
 * Copyright (c) 2015-2016 Intel Corporation
 *
 * SPDX-License-Identifier: Apache-2.0
 */

#include <zephyr/types.h>
#include <stddef.h>
#include <sys/printk.h>
#include <sys/util.h>

#include <bluetooth/bluetooth.h>
#include <bluetooth/hci.h>

#define DEVICE_NAME CONFIG_BT_DEVICE_NAME
#define DEVICE_NAME_LEN (sizeof(DEVICE_NAME) - 1)

/*
 * Set Advertisement data. Based on the Eddystone specification:
 * https://github.com/google/eddystone/blob/master/protocol-specification.md
 * https://github.com/google/eddystone/tree/master/eddystone-url
 */

static const struct bt_data ad[] = {
    BT_DATA_BYTES(BT_DATA_FLAGS, BT_LE_AD_NO_BREDR),
    BT_DATA_BYTES(BT_DATA_UUID16_ALL, 0xaa, 0xfe),
    BT_DATA_BYTES(BT_DATA_SVC_DATA16,
                  0xaa, 0xfe,
                  0x00,
                  'H','e','l','l','o',' ',
                  'W','o','r','l','d',' ',
                  'b','y',' ','B','L','E')
};

/* Set Scan Response data */
static const struct bt_data sd[] = {
    BT_DATA(BT_DATA_NAME_COMPLETE, DEVICE_NAME, DEVICE_NAME_LEN),
};

static void bt_ready(int err)
{
```

```
    char addr_s[BT_ADDR_LE_STR_LEN];
    bt_addr_le_t addr = {0};
    size_t count = 1;

    if (err) {
       printk("Bluetooth init failed (err %d)\n", err);
       return;
    }

    printk("Bluetooth initialized\n");

    /* Start advertising */
    err = bt_le_adv_start(BT_LE_ADV_NCONN_IDENTITY, ad, ARRAY_SIZE(ad),
                          sd, ARRAY_SIZE(sd));

    if (err) {
       printk("Advertising failed to start (err %d)\n", err);
       return;
    }


    /* For connectable advertising you would use
     * bt_le_oob_get_local().    For non-connectable non-identity
     * advertising an non-resolvable private address is used;
     * there is no API to retrieve that.
     */

    bt_id_get(&addr, &count);
    bt_addr_le_to_str(&addr, addr_s, sizeof(addr_s));

    printk("Beacon started, advertising as %s\n", addr_s);
}

void main(void)
{
    int err;

    //printk("Starting Beacon Demo\n");
    printk("Beacon Service\n");

    /* Initialize the Bluetooth Subsystem */
    err = bt_enable(bt_ready);
    if (err) {
       printk("Bluetooth init failed (err %d)\n", err);
    }
}
```

Central

```
/*
 * Copyright (c) 2020 Nordic Semiconductor ASA
 *
 * SPDX-License-Identifier: LicenseRef-Nordic-5-Clause
 */

/*
 * ReadMe_Central
 * |Overview
 * Zephyr OS 異質物聯網系統之 Central 實現，接收 Ble_Beacon(BLE)與 Light_switch(ZigBee)所傳之訊號，
 *  並顯示其 Address、距離與 RSSI
 * |Main Skill
 * d = pow(10,(abs(rssi)-56)/20.0) //距離換算
 * zboss_signal_handler(param) //ZBOSS 訊號處理
 * zb_get_app_signal(bufid, &sg_p) //解包訊號
 * ZB_GET_APP_SIGNAL_STATUS(bufid) //獲取訊號狀態
 * zb_zdo_get_diag_data(address, lqi, rssi) //獲取 ZigBee 之 LQI 與 RSSI
 * device_found(addr, rssi, type, ad) //獲取 BLE 訊號
 * |Building and Running
 * cd Central
 * west build -b nrf52840dk_nrf52840
 * west flash
 * sudo minicom -D /dev/<your device>
 *  其發送狀況可於 Ubuntu 終端機呈現
 */

/** @file
 *   @brief Simple Zigbee network coordinator implementation
 */

#include <zephyr.h>
#include <device.h>
#include <logging/log.h>
#include <dk_buttons_and_leds.h>

#include <zboss_api.h>
#include <zb_mem_config_max.h>
#include <zigbee/zigbee_error_handler.h>
#include <zigbee/zigbee_app_utils.h>
#include <zb_nrf_platform.h>

//
#include <zephyr/types.h>
#include <stddef.h>
#include <errno.h>
#include <sys/printk.h>

#include <bluetooth/bluetooth.h>
#include <bluetooth/hci.h>
#include <bluetooth/conn.h>
#include <bluetooth/uuid.h>
#include <bluetooth/gatt.h>
#include <sys/byteorder.h>

#include <stdio.h>
#include <stdlib.h>
```

```
#include <math.h>
#include <sys/util.h>
//

#define RUN_STATUS_LED                  DK_LED1
#define RUN_LED_BLINK_INTERVAL    1000

/* LED indicating that network is opened for new nodes */
#define ZIGBEE_NETWORK_STATE_LED                DK_LED3
#define BLE_STATE_LED                                   DK_LED4

/* Button which reopens the Zigbee Network */
#define KEY_ZIGBEE_NETWORK_REOPEN               DK_BTN1_MSK
#define KEY_BLE_REOPEN                                  DK_BTN2_MSK

/**
 * If set to ZB_TRUE then device will not open the network
 * after forming or reboot.
 */
#define ZIGBEE_MANUAL_STEERING                  ZB_FALSE

#define ZIGBEE_PERMIT_LEGACY_DEVICES          ZB_FALSE

#ifndef ZB_COORDINATOR_ROLE
#error Define ZB_COORDINATOR_ROLE to compile coordinator source code.
#endif

LOG_MODULE_REGISTER(app);

//
static void start_scan(void);
static struct bt_conn *default_conn;

/**@brief Callback used in order to visualise network steering period.
 *
 * @param[in]   param   Not used. Required by callback type definition.
 */
static void steering_finished(zb_uint8_t param)
{
    ARG_UNUSED(param);
    LOG_INF("Network steering finished");
    dk_set_led_off(ZIGBEE_NETWORK_STATE_LED);
    (void)(ZB_SCHEDULE_APP_ALARM_CANCEL(
            steering_finished, ZB_ALARM_ANY_PARAM));
}

/**@brief Callback for button events.
 *
 * @param[in]   button_state  Bitmask containing buttons state.
 * @param[in]   has_changed   Bitmask containing buttons
 *                            that has changed their state.
 */
static void button_changed(uint32_t button_state, uint32_t has_changed)
{
    /* Calculate bitmask of buttons that are pressed
     * and have changed their state.
     */
    uint32_t buttons = button_state & has_changed;
```

```
    zb_bool_t comm_status;

    if (buttons & KEY_ZIGBEE_NETWORK_REOPEN) {
      (void)(ZB_SCHEDULE_APP_ALARM_CANCEL(
            steering_finished, ZB_ALARM_ANY_PARAM));

      comm_status = bdb_start_top_level_commissioning(
            ZB_BDB_NETWORK_STEERING);
      if (comm_status) {
            LOG_INF("Top level comissioning restated");
      } else {
            LOG_INF("Top level comissioning hasn't finished yet!");
      }
    }
}

static void device_found(const bt_addr_le_t *addr, int8_t rssi, uint8_t type,
              struct net_buf_simple *ad)
{
    char addr_str[BT_ADDR_LE_STR_LEN];

    if (default_conn)
    {
      return;
    }

    /* We're only interested in connectable events */
    if (type != BT_GAP_ADV_TYPE_ADV_SCAN_IND)
    {
      return;
    }

    bt_addr_le_to_str(addr, addr_str, sizeof(addr_str));

    /* connect only to devices in close proximity */
    if (rssi < -80)
    {
      return;
    }

    float d = pow(10,(abs(rssi)-56)/20.0);
    printk("BLE found: %s ,within %.2f m radius (RSSI %d)\n", addr_str, d, rssi);
    if(ad->len>=12)
    {
      printk("BLE data: ");
      for(int i=12;i<ad->len;i++)
      {
            printk("%c", ad->data[i]);
      }
      printk("\n");
    }

    if (bt_le_scan_stop())
    {
      return;
    }
    if(true)
    {
```

```
        k_sleep(K_MSEC(1000));
        //start_scan();
    }
    start_scan();
}

static void start_scan(void)
{
    int err;

    /* This demo doesn't require active scan */
    err = bt_le_scan_start(BT_LE_SCAN_PASSIVE, device_found);
    if (err) {
        printk("Scanning failed to start (err %d)\n", err);
        return;
    }
    //start_scan();

    printk("Scanning successfully started\n");
}

/**@brief Function for initializing LEDs and Buttons. */
static void configure_gpio(void)
{
    int err;

    err = dk_buttons_init(button_changed);
    if (err) {
        LOG_ERR("Cannot init buttons (err: %d)", err);
    }

    err = dk_leds_init();
    if (err) {
        LOG_ERR("Cannot init LEDs (err: %d)", err);
    }
}

/**@brief Zigbee stack event handler.
 *
 * @param[in]   bufid   Reference to the Zigbee stack buffer
 *                      used to pass signal.
 */
void zboss_signal_handler(zb_bufid_t bufid)
{
    /* Read signal description out of memory buffer. */
    zb_zdo_app_signal_hdr_t *sg_p = NULL;
    zb_zdo_app_signal_type_t sig = zb_get_app_signal(bufid, &sg_p);
    zb_ret_t status = ZB_GET_APP_SIGNAL_STATUS(bufid);
    zb_ret_t zb_err_code;
    zb_bool_t comm_status;
    zb_time_t timeout_bi;
    zb_uint8_t lqi;
    zb_int8_t rssi;

    switch (sig) {
    case ZB_BDB_SIGNAL_DEVICE_REBOOT:
        /* BDB initialization completed after device reboot,
         * use NVRAM contents during initialization.
```

```
        * Device joined/rejoined and started.
        */
      if (status == RET_OK) {
            if (ZIGBEE_MANUAL_STEERING == ZB_FALSE) {
                  LOG_INF("Start network steering");
                  comm_status = bdb_start_top_level_commissioning(
                        ZB_BDB_NETWORK_STEERING);
                  ZB_COMM_STATUS_CHECK(comm_status);
            } else {
                  LOG_INF("Coordinator restarted successfully");
            }
      } else {
            LOG_ERR("Failed to initialize Zigbee stack using NVRAM data (status: %d)",
                  status);
      }
      break;

   case ZB_BDB_SIGNAL_STEERING:
      if (status == RET_OK) {
            if (ZIGBEE_PERMIT_LEGACY_DEVICES == ZB_TRUE) {
                  LOG_INF("Allow pre-Zigbee 3.0 devices to join the network");
                  zb_bdb_set_legacy_device_support(1);
            }

            /* Schedule an alarm to notify about the end
              * of steering period
              */
            LOG_INF("Network steering started");
            zb_err_code = ZB_SCHEDULE_APP_ALARM(
                  steering_finished, 0,
                  ZB_TIME_ONE_SECOND *3);
            ZB_ERROR_CHECK(zb_err_code);
      }
      break;

   case ZB_ZDO_SIGNAL_DEVICE_ANNCE: {
      zb_zdo_signal_device_annce_params_t *dev_annce_params =
            ZB_ZDO_SIGNAL_GET_PARAMS(
                  sg_p, zb_zdo_signal_device_annce_params_t);

      LOG_INF("New device commissioned or rejoined (short: 0x%04hx)",
            dev_annce_params->device_short_addr);
      zb_zdo_get_diag_data(dev_annce_params->device_short_addr,&lqi,&rssi);
   float d = pow(10,(abs(rssi)-56)/20.0);
   printk("ZigBee found: 0x%04hx ,", dev_annce_params->device_short_addr);
   printk("within %.2f m radius (RSSI %d)\n", d, rssi);
   printk("ZigBee info: %p\n",zb_buf_get_tail(bufid,sizeof(bufid)));

      zb_err_code = ZB_SCHEDULE_APP_ALARM_CANCEL(steering_finished,
                                          ZB_ALARM_ANY_PARAM);
      if (zb_err_code == RET_OK) {
            LOG_INF("Joining period extended.");
            zb_err_code = ZB_SCHEDULE_APP_ALARM(
                  steering_finished, 0,
                  ZB_TIME_ONE_SECOND *3);
            ZB_ERROR_CHECK(zb_err_code);
      }
   } break;
```

```
    default:
      /* Call default signal handler. */
      ZB_ERROR_CHECK(zigbee_default_signal_handler(bufid));
      break;
    }

    /* Update network status LED */
    /*if (ZB_JOINED() &&
          (ZB_SCHEDULE_GET_ALARM_TIME(steering_finished, ZB_ALARM_ANY_PARAM,
                        &timeout_bi) == RET_OK)) {
      //printk("on\n");
      dk_set_led_on(ZIGBEE_NETWORK_STATE_LED);
      dk_set_led_off(BLE_STATE_LED);
    } else {
      //printk("off\n");
      dk_set_led_off(ZIGBEE_NETWORK_STATE_LED);
      dk_set_led_on(BLE_STATE_LED);
    }*/

    /*
     * All callbacks should either reuse or free passed buffers.
     * If bufid == 0, the buffer is invalid (not passed).
     */
    if (bufid) {
      zb_buf_free(bufid);
    }
}

void error(void)
{
    dk_set_leds_state(DK_ALL_LEDS_MSK, DK_NO_LEDS_MSK);

    while (true) {
      /* Spin for ever */
      k_sleep(K_MSEC(1000));
    }
}

void main(void)
{
    int blink_status = 0;
    int err;
    int k=0;

    LOG_INF("Starting ZBOSS Coordinator example");
    printk("Scanning successfully started\n");

    /* Initialize */
    err = bt_enable(NULL);
    if (err) {
      printk("Bluetooth init failed (err %d)\n", err);
      return;
    }
    configure_gpio();

    /* Start Zigbee default thread */
    zigbee_enable();
```

```
    //bt_conn_cb_register(&conn_callbacks);
    start_scan();

    LOG_INF("ZBOSS Coordinator example started");

    while (1) {
      ++k;
      dk_set_led(RUN_STATUS_LED, (++blink_status) % 2);
      k_sleep(K_MSEC(RUN_LED_BLINK_INTERVAL));
      if(k%2==0)
      {
            dk_set_led_on(ZIGBEE_NETWORK_STATE_LED);
            dk_set_led_off(BLE_STATE_LED);
      }
      else
      {
            dk_set_led_off(ZIGBEE_NETWORK_STATE_LED);
            dk_set_led_on(BLE_STATE_LED);
      }
    }
}
```

light_switch

```
/*
 * Copyright (c) 2020 Nordic Semiconductor ASA
 *
 * SPDX-License-Identifier: LicenseRef-Nordic-5-Clause
 */

/*
 * ReadMe_light_switch
 * |Overview
 *  為 Simples 所提供之檔案，本專案用於測試 ZigBee 之 Beacon 的角色
 * |Building and Running
 * cd Light_switch
 * west build -b nrf52840dk_nrf52840
 * west flash
 * sudo minicom -D /dev/<your device>
 *  其發送狀況可於 Ubuntu 終端機呈現
 */

/** @file
 * @brief Dimmer switch for HA profile implementation.
 */

#include <zephyr.h>
#include <device.h>
#include <logging/log.h>
#include <dk_buttons_and_leds.h>
```

```
#include <ram_pwrdn.h>

#include <zboss_api.h>
#include <zboss_api_addons.h>
#include <zigbee/zigbee_app_utils.h>
#include <zigbee/zigbee_error_handler.h>
#include <zb_nrf_platform.h>
#include "zb_mem_config_custom.h"

#if CONFIG_ZIGBEE_FOTA
#include <zigbee/zigbee_fota.h>
#include <sys/reboot.h>
#include <dfu/mcuboot.h>

/* LED indicating OTA Client Activity. */
/*test for zigbee*/
#define OTA_ACTIVITY_LED                DK_LED2
#endif /* CONFIG_ZIGBEE_FOTA */

#if CONFIG_BT_NUS
#include "nus_cmd.h"

/* LED which indicates that Central is connected. */
#define NUS_STATUS_LED                  DK_LED1
/* UART command that will turn on found light bulb(s). */
#define COMMAND_ON                      "n"
/**< UART command that will turn off found light bulb(s). */
#define COMMAND_OFF                     "f"
/**< UART command that will turn toggle found light bulb(s). */
#define COMMAND_TOGGLE                  "t"
/**< UART command that will increase brightness of found light bulb(s). */
#define COMMAND_INCREASE                "i"
/**< UART command that will decrease brightness of found light bulb(s). */
#define COMMAND_DECREASE                "d"
#endif /* CONFIG_BT_NUS */

/* Source endpoint used to control light bulb. */
#define LIGHT_SWITCH_ENDPOINT           1
/* Delay between the light switch startup and light bulb finding procedure. */
#define MATCH_DESC_REQ_START_DELAY K_SECONDS(2)
/* Timeout for finding procedure. */
#define MATCH_DESC_REQ_TIMEOUT          K_SECONDS(5)
/* Find only non-sleepy device. */
#define MATCH_DESC_REQ_ROLE             ZB_NWK_BROADCAST_RX_ON_WHEN_IDLE

/* Do not erase NVRAM to save the network parameters after device reboot or
 * power-off. NOTE: If this option is set to ZB_TRUE then do full device erase
 * for all network devices before running other samples.
 */
#define ERASE_PERSISTENT_CONFIG         ZB_FALSE
/* LED indicating that light switch successfully joind Zigbee network. */
#define ZIGBEE_NETWORK_STATE_LED        DK_LED3
/* LED indicating that light witch found a light bulb to control. */
#define BULB_FOUND_LED                  DK_LED4
/* Button ID used to switch on the light bulb. */
#define BUTTON_ON                       DK_BTN1_MSK
/* Button ID used to switch off the light bulb. */
#define BUTTON_OFF                      DK_BTN2_MSK
```

```
/* Dim step size - increases/decreses current level (range 0x000 - 0xfe). */
#define DIMM_STEP                       15
/* Button ID used to enable sleepy behavior. */
#define BUTTON_SLEEPY                   DK_BTN3_MSK

/* Transition time for a single step operation in 0.1 sec units.
 * 0xFFFF - immediate change.
 */
#define DIMM_TRANSACTION_TIME           2

/* Time after which the button state is checked again to detect button hold,
 * the dimm command is sent again.
 */
#define BUTTON_LONG_POLL_TMO            K_MSEC(500)

#if !defined ZB_ED_ROLE
#error Define ZB_ED_ROLE to compile light switch (End Device) source code.
#endif

LOG_MODULE_REGISTER(app);

struct bulb_context {
        zb_uint8_t      endpoint;
        zb_uint16_t     short_addr;
        struct k_timer find_alarm;
};

struct buttons_context {
        uint32_t        state;
        atomic_t        long_poll;
        struct k_timer alarm;
};

static struct bulb_context bulb_ctx;
static struct buttons_context buttons_ctx;
static zb_uint8_t   attr_zcl_version = ZB_ZCL_VERSION;
static zb_uint8_t   attr_power_source = ZB_ZCL_BASIC_POWER_SOURCE_UNKNOWN;
static zb_uint16_t attr_identify_time;

/* Declare attribute list for Basic cluster. */
ZB_ZCL_DECLARE_BASIC_ATTRIB_LIST(basic_attr_list, &attr_zcl_version,
                                 &attr_power_source);

/* Declare attribute list for Identify cluster. */
ZB_ZCL_DECLARE_IDENTIFY_ATTRIB_LIST(identify_attr_list, &attr_identify_time);

/* Declare cluster list for Dimmer Switch device (Identify, Basic, Scenes,
 * Groups, On Off, Level Control).
 * Only clusters Identify and Basic have attributes.
 */
ZB_HA_DECLARE_DIMMER_SWITCH_CLUSTER_LIST(dimmer_switch_clusters,
                                         basic_attr_list,
                                         identify_attr_list);

/* Declare endpoint for Dimmer Switch device. */
ZB_HA_DECLARE_DIMMER_SWITCH_EP(dimmer_switch_ep,
                               LIGHT_SWITCH_ENDPOINT,
                               dimmer_switch_clusters);
```

```
/* Declare application's device context (list of registered endpoints)
 * for Dimmer Switch device.
 */
#ifndef CONFIG_ZIGBEE_FOTA
ZBOSS_DECLARE_DEVICE_CTX_1_EP(dimmer_switch_ctx, dimmer_switch_ep);
#else

   #if LIGHT_SWITCH_ENDPOINT == CONFIG_ZIGBEE_FOTA_ENDPOINT
      #error "Light switch and Zigbee OTA endpoints should be different."
   #endif

extern zb_af_endpoint_desc_t zigbee_fota_client_ep;
ZBOSS_DECLARE_DEVICE_CTX_2_EP(dimmer_switch_ctx,
                              zigbee_fota_client_ep,
                              dimmer_switch_ep);
#endif /* CONFIG_ZIGBEE_FOTA */

/* Forward declarations. */
static void light_switch_button_handler(struct k_timer *timer);
static void find_light_bulb_alarm(struct k_timer *timer);
static void find_light_bulb(zb_bufid_t bufid);
static void light_switch_send_on_off(zb_bufid_t bufid, zb_uint16_t on_off);


/**@brief Callback for button events.
 *
 * @param[in]   button_state  Bitmask containing buttons state.
 * @param[in]   has_changed   Bitmask containing buttons that has
 *                            changed their state.
 */
static void button_handler(uint32_t button_state, uint32_t has_changed)
{
	zb_uint16_t cmd_id;
	zb_ret_t zb_err_code;

	/* Inform default signal handler about user input at the device. */
	user_input_indicate();

	if (bulb_ctx.short_addr == 0xFFFF) {
		LOG_DBG("No bulb found yet.");
		return;
	}

	switch (has_changed) {
	case BUTTON_ON:
		LOG_DBG("ON - button changed");
		cmd_id = ZB_ZCL_CMD_ON_OFF_ON_ID;
		break;
	case BUTTON_OFF:
		LOG_DBG("OFF - button changed");
		cmd_id = ZB_ZCL_CMD_ON_OFF_OFF_ID;
		break;
	default:
		LOG_DBG("Unhandled button");
		return;
	}
```

```
	switch (button_state) {
	case BUTTON_ON:
	case BUTTON_OFF:
		LOG_DBG("Button pressed");
		buttons_ctx.state = button_state;
		/* Alarm can be scheduled only once. Next alarm only resets
		 * counting.
		 */
		k_timer_start(&buttons_ctx.alarm, BUTTON_LONG_POLL_TMO,
				K_NO_WAIT);
		break;
	case 0:
		LOG_DBG("Button released");
		k_timer_stop(&buttons_ctx.alarm);
		if (atomic_set(&buttons_ctx.long_poll, ZB_FALSE) == ZB_FALSE) {
			/* Allocate output buffer and send on/off command. */
			zb_err_code = zb_buf_get_out_delayed_ext(
				light_switch_send_on_off, cmd_id, 0);
			ZB_ERROR_CHECK(zb_err_code);
		}
	}
}

/**@brief Function for initializing LEDs and Buttons. */
static void configure_gpio(void)
{
	int err;

	err = dk_buttons_init(button_handler);
	if (err) {
		LOG_ERR("Cannot init buttons (err: %d)", err);
	}

	err = dk_leds_init();
	if (err) {
		LOG_ERR("Cannot init LEDs (err: %d)", err);
	}
}

static void alarm_timers_init(void)
{
	k_timer_init(&buttons_ctx.alarm, light_switch_button_handler, NULL);
	k_timer_init(&bulb_ctx.find_alarm, find_light_bulb_alarm, NULL);
}

/**@brief Function for sending ON/OFF requests to the light bulb.
 *
 * @param[in]	bufid	Non-zero reference to Zigbee stack buffer that will be
 *			used to construct on/off request.
 * @param[in]	cmd_id	ZCL command id.
 */
static void light_switch_send_on_off(zb_bufid_t bufid, zb_uint16_t cmd_id)
{
	LOG_INF("Send ON/OFF command: %d", cmd_id);

	ZB_ZCL_ON_OFF_SEND_REQ(bufid,
				bulb_ctx.short_addr,
				ZB_APS_ADDR_MODE_16_ENDP_PRESENT,
```

```
                        bulb_ctx.endpoint,
                        LIGHT_SWITCH_ENDPOINT,
                        ZB_AF_HA_PROFILE_ID,
                        ZB_ZCL_DISABLE_DEFAULT_RESPONSE,
                        cmd_id,
                        NULL);
}

/**@brief Function for sending step requests to the light bulb.
 *
 * @param[in]   bufid           Non-zero reference to Zigbee stack buffer that
 *                                will be used to construct step request.
 * @param[in]   cmd_id          ZCL command id.
 */
static void light_switch_send_step(zb_bufid_t bufid, zb_uint16_t cmd_id)
{
    LOG_INF("Send step level command: %d", cmd_id);

    ZB_ZCL_LEVEL_CONTROL_SEND_STEP_REQ(bufid,
                                       bulb_ctx.short_addr,
                                       ZB_APS_ADDR_MODE_16_ENDP_PRESENT,
                                       bulb_ctx.endpoint,
                                       LIGHT_SWITCH_ENDPOINT,
                                       ZB_AF_HA_PROFILE_ID,
                                       ZB_ZCL_DISABLE_DEFAULT_RESPONSE,
                                       NULL,
                                       cmd_id,
                                       DIMM_STEP,
                                       DIMM_TRANSACTION_TIME);
}

/**@brief Callback function receiving finding procedure results.
 *
 * @param[in]   bufid   Reference to Zigbee stack buffer used to pass
 *                        received data.
 */
static void find_light_bulb_cb(zb_bufid_t bufid)
{
    /* Get the beginning of the response. */
    zb_zdo_match_desc_resp_t *resp =
                        (zb_zdo_match_desc_resp_t *) zb_buf_begin(bufid);
    /* Get the pointer to the parameters buffer, which stores APS layer
     * response.
     */
    zb_apsde_data_indication_t *ind   = ZB_BUF_GET_PARAM(bufid,
                                    zb_apsde_data_indication_t);
    zb_uint8_t *match_ep;

    if ((resp->status == ZB_ZDP_STATUS_SUCCESS) &&
        (resp->match_len > 0) &&
        (bulb_ctx.short_addr == 0xFFFF)) {

        /* Match EP list follows right after response header. */
        match_ep = (zb_uint8_t *)(resp + 1);

        /* We are searching for exact cluster, so only 1 EP
         * may be found.
         */
```

```
		bulb_ctx.endpoint    = *match_ep;
		bulb_ctx.short_addr = ind->src_addr;

		LOG_INF("Found bulb addr: %d ep: %d",
			bulb_ctx.short_addr,
			bulb_ctx.endpoint);

		k_timer_stop(&bulb_ctx.find_alarm);
		dk_set_led_on(BULB_FOUND_LED);
	} else {
		LOG_INF("Bulb not found, try again");
	}

	if (bufid) {
		zb_buf_free(bufid);
	}
}

/**@brief Find bulb allarm handler.
 *
 * @param[in]	timer	Address of timer.
 */
static void find_light_bulb_alarm(struct k_timer *timer)
{
	ZB_ERROR_CHECK(zb_buf_get_out_delayed(find_light_bulb));
}

/**@brief Function for sending ON/OFF and Level Control find request.
 *
 * @param[in]	bufid	Reference to Zigbee stack buffer that will be used to
 *                      construct find request.
 */
static void find_light_bulb(zb_bufid_t bufid)
{
	zb_zdo_match_desc_param_t *req;

	/* Initialize pointers inside buffer and reserve space for
	 * zb_zdo_match_desc_param_t request.
	 */
	req = zb_buf_initial_alloc(bufid,
		sizeof(zb_zdo_match_desc_param_t) + (1) * sizeof(zb_uint16_t));

	req->nwk_addr		= MATCH_DESC_REQ_ROLE;
	req->addr_of_interest = MATCH_DESC_REQ_ROLE;
	req->profile_id		= ZB_AF_HA_PROFILE_ID;

	/* We are searching for 2 clusters: On/Off and Level Control Server. */
	req->num_in_clusters	= 2;
	req->num_out_clusters = 0;
	req->cluster_list[0]	= ZB_ZCL_CLUSTER_ID_ON_OFF;
	req->cluster_list[1]	= ZB_ZCL_CLUSTER_ID_LEVEL_CONTROL;

	/* Set 0xFFFF to reset short address in order to parse
	 * only one response.
	 */
	bulb_ctx.short_addr = 0xFFFF;
	(void)zb_zdo_match_desc_req(bufid, find_light_bulb_cb);
}
```

```
/**@brief Callback for detecting button press duration.
 *
 * @param[in]   timer   Address of timer.
 */
static void light_switch_button_handler(struct k_timer *timer)
{
	zb_ret_t zb_err_code;
	zb_uint16_t cmd_id;

	if (dk_get_buttons() & buttons_ctx.state) {
		atomic_set(&buttons_ctx.long_poll, ZB_TRUE);
		if (buttons_ctx.state == BUTTON_ON) {
			cmd_id = ZB_ZCL_LEVEL_CONTROL_STEP_MODE_UP;
		} else {
			cmd_id = ZB_ZCL_LEVEL_CONTROL_STEP_MODE_DOWN;
		}

		/* Allocate output buffer and send step command. */
		zb_err_code = zb_buf_get_out_delayed_ext(light_switch_send_step,
							 cmd_id,
							 0);
		if (!zb_err_code) {
			LOG_WRN("Buffer is full");
		}

		k_timer_start(&buttons_ctx.alarm, BUTTON_LONG_POLL_TMO,
				K_NO_WAIT);
	} else {
		atomic_set(&buttons_ctx.long_poll, ZB_FALSE);
	}
}

#ifdef CONFIG_ZIGBEE_FOTA
static void confirm_image(void)
{
	if (!boot_is_img_confirmed()) {
		int ret = boot_write_img_confirmed();

		if (ret) {
			LOG_ERR("Couldn't confirm image: %d", ret);
		} else {
			LOG_INF("Marked image as OK");
		}
	}
}

static void ota_evt_handler(const struct zigbee_fota_evt *evt)
{
	switch (evt->id) {
	case ZIGBEE_FOTA_EVT_PROGRESS:
		dk_set_led(OTA_ACTIVITY_LED, evt->dl.progress % 2);
		break;

	case ZIGBEE_FOTA_EVT_FINISHED:
		LOG_INF("Reboot application.");
		sys_reboot(SYS_REBOOT_COLD);
		break;
```

```
	case ZIGBEE_FOTA_EVT_ERROR:
		LOG_ERR("OTA image transfer failed.");
		break;

	default:
		break;
	}
}
#endif /* CONFIG_ZIGBEE_FOTA */

/**@brief Zigbee stack event handler.
 *
 * @param[in]	bufid	Reference to the Zigbee stack buffer
 *			used to pass signal.
 */
void zboss_signal_handler(zb_bufid_t bufid)
{
	zb_zdo_app_signal_hdr_t	*sig_hndler = NULL;
	zb_zdo_app_signal_type_t	sig = zb_get_app_signal(bufid, &sig_hndler);
	zb_ret_t			status = ZB_GET_APP_SIGNAL_STATUS(bufid);

	/* Update network status LED. */
	zigbee_led_status_update(bufid, ZIGBEE_NETWORK_STATE_LED);

#ifdef CONFIG_ZIGBEE_FOTA
	/* Pass signal to the OTA client implementation. */
	zigbee_fota_signal_handler(bufid);
#endif /* CONFIG_ZIGBEE_FOTA */

	switch (sig) {
	case ZB_BDB_SIGNAL_DEVICE_REBOOT:
	/* fall-through */
	case ZB_BDB_SIGNAL_STEERING:
		/* Call default signal handler. */
		ZB_ERROR_CHECK(zigbee_default_signal_handler(bufid));
		if (status == RET_OK) {
			/* Check the light device address. */
			if (bulb_ctx.short_addr == 0xFFFF) {
				k_timer_start(&bulb_ctx.find_alarm,
					      MATCH_DESC_REQ_START_DELAY,
					      MATCH_DESC_REQ_TIMEOUT);
			}
		}
		break;
	default:
		/* Call default signal handler. */
		ZB_ERROR_CHECK(zigbee_default_signal_handler(bufid));
		break;
	}

	if (bufid) {
		zb_buf_free(bufid);
	}
}

#if CONFIG_BT_NUS
```

```
static void turn_on_cmd(struct k_work *item)
{
	ARG_UNUSED(item);
	zb_buf_get_out_delayed_ext(light_switch_send_on_off,
				   ZB_ZCL_CMD_ON_OFF_ON_ID, 0);
}

static void turn_off_cmd(struct k_work *item)
{
	ARG_UNUSED(item);
	zb_buf_get_out_delayed_ext(light_switch_send_on_off,
				   ZB_ZCL_CMD_ON_OFF_OFF_ID, 0);
}

static void toggle_cmd(struct k_work *item)
{
	ARG_UNUSED(item);
	zb_buf_get_out_delayed_ext(light_switch_send_on_off,
				   ZB_ZCL_CMD_ON_OFF_TOGGLE_ID, 0);
}

static void increase_cmd(struct k_work *item)
{
	ARG_UNUSED(item);
	zb_buf_get_out_delayed_ext(light_switch_send_step,
				   ZB_ZCL_LEVEL_CONTROL_STEP_MODE_UP, 0);
}

static void decrease_cmd(struct k_work *item)
{
	ARG_UNUSED(item);
	zb_buf_get_out_delayed_ext(light_switch_send_step,
				   ZB_ZCL_LEVEL_CONTROL_STEP_MODE_DOWN, 0);
}

static void on_nus_connect(struct k_work *item)
{
	ARG_UNUSED(item);
	dk_set_led_on(NUS_STATUS_LED);
}

static void on_nus_disconnect(struct k_work *item)
{
	ARG_UNUSED(item);
	dk_set_led_off(NUS_STATUS_LED);
}

static struct nus_entry commands[] = {
	NUS_COMMAND(COMMAND_ON, turn_on_cmd),
	NUS_COMMAND(COMMAND_OFF, turn_off_cmd),
	NUS_COMMAND(COMMAND_TOGGLE, toggle_cmd),
	NUS_COMMAND(COMMAND_INCREASE, increase_cmd),
	NUS_COMMAND(COMMAND_DECREASE, decrease_cmd),
	NUS_COMMAND(NULL, NULL),
};

#endif /* CONFIG_BT_NUS */
```

```
void main(void)
{
	LOG_INF("Starting ZBOSS Light Switch example");

	/* Initialize. */
	configure_gpio();
	alarm_timers_init();

	zigbee_erase_persistent_storage(ERASE_PERSISTENT_CONFIG);
	zb_set_ed_timeout(ED_AGING_TIMEOUT_64MIN);
	zb_set_keepalive_timeout(ZB_MILLISECONDS_TO_BEACON_INTERVAL(3000));

	/* Set default bulb short_addr. */
	bulb_ctx.short_addr = 0xFFFF;

	/* If "sleepy button" is defined, check its state during Zigbee
	 * initialization and enable sleepy behavior at device if defined button
	 * is pressed. Additionally, power off unused sections of RAM to lower
	 * device power consumption.
	 */
#if defined BUTTON_SLEEPY
	if (dk_get_buttons() & BUTTON_SLEEPY) {
		zigbee_configure_sleepy_behavior(true);

		if (IS_ENABLED(CONFIG_RAM_POWER_DOWN_LIBRARY)) {
			power_down_unused_ram();
		}
	}
#endif

#ifdef CONFIG_ZIGBEE_FOTA
	/* Initialize Zigbee FOTA download service. */
	zigbee_fota_init(ota_evt_handler);

	/* Mark the current firmware as valid. */
	confirm_image();

	/* Register callback for handling ZCL commands. */
	ZB_ZCL_REGISTER_DEVICE_CB(zigbee_fota_zcl_cb);
#endif /* CONFIG_ZIGBEE_FOTA */

	/* Register dimmer switch device context (endpoints). */
	ZB_AF_REGISTER_DEVICE_CTX(&dimmer_switch_ctx);

	/* Start Zigbee default thread. */
	zigbee_enable();

#if CONFIG_BT_NUS
	/* Initalize NUS command service. */
	nus_cmd_init(on_nus_connect, on_nus_disconnect, commands);
#endif /* CONFIG_BT_NUS */

	LOG_INF("ZBOSS Light Switch example started");

	while (1) {
		k_sleep(K_FOREVER);
	}
}
```