How to build the simplest nRF52 BLE Peripheral application (Lightbulb use case)

By Mohammad Afaneh (https://www.novelbits.io/author/mafaneh/) | May 29, 2018 | 20 (https://www.novelbits.io/smart-ble-lightbulb-application-nrf52/#comments)

Let's face it…

One of the hardest things when working with BLE is **simply getting started.**

Whether it's the setup of the IDE, the configuration of the project, or the implementation the BLE application.

I've been there… I've felt lost, not knowing where and how to start… This is (especially true since I was trying to learn the technology itself (BLE) in addition to learning an SDK, platform, and IDE all at once! It just felt overwhelming and way too many things to learn at one time.

Lately, I've been focusing on one platform/chipset: Nordic's nRF52. This is due to one main reason: **I've found it to be the most developer-friendly platform out there.** *(It also helps that you get a FREE commercial license for a professional IDE: Segger Embedded Studio (SES)!)*

There's nothing wrong with the other platforms and chipsets, but it also helps if you stick to one platform that you feel comfortable with (at least for a given period of time, especially in the beginning of your journey in learning a technology).

Ok, enough with the rant, and let's get into what this post is all about:

To guide you through setting up and developing the most basic BLE peripheral application: a smart BLE lightbulb application you can control from your smartphone.

In the previous blog post (The complete cross-platform nRF development tutorial (https://www.novelbits.io/cross-platform-nrf-development-tutorial/)), we went over how to set up the IDE of choice for developing nRF52 applications (Segger Embedded Studio) on any platform (macOS, Windows, Linux). In this post, we'll focus on **developing the BLE peripheral application**, building it, debugging it, and finally testing it from a mobile phone application.

We're going to build a very simple BLE lightbulb application that allows you to **turn ON/OFF** an **LED** on the nRF52840 development kit. The Peripheral application will also expose the **battery level** of a coin-cell battery installed in the development kit. The peripheral will notify the Central (mobile phone application in our case) when the battery level gets updated.

Keeping the example dead-simple is extremely important. You can always customize and expand your application once you've learned the basics.

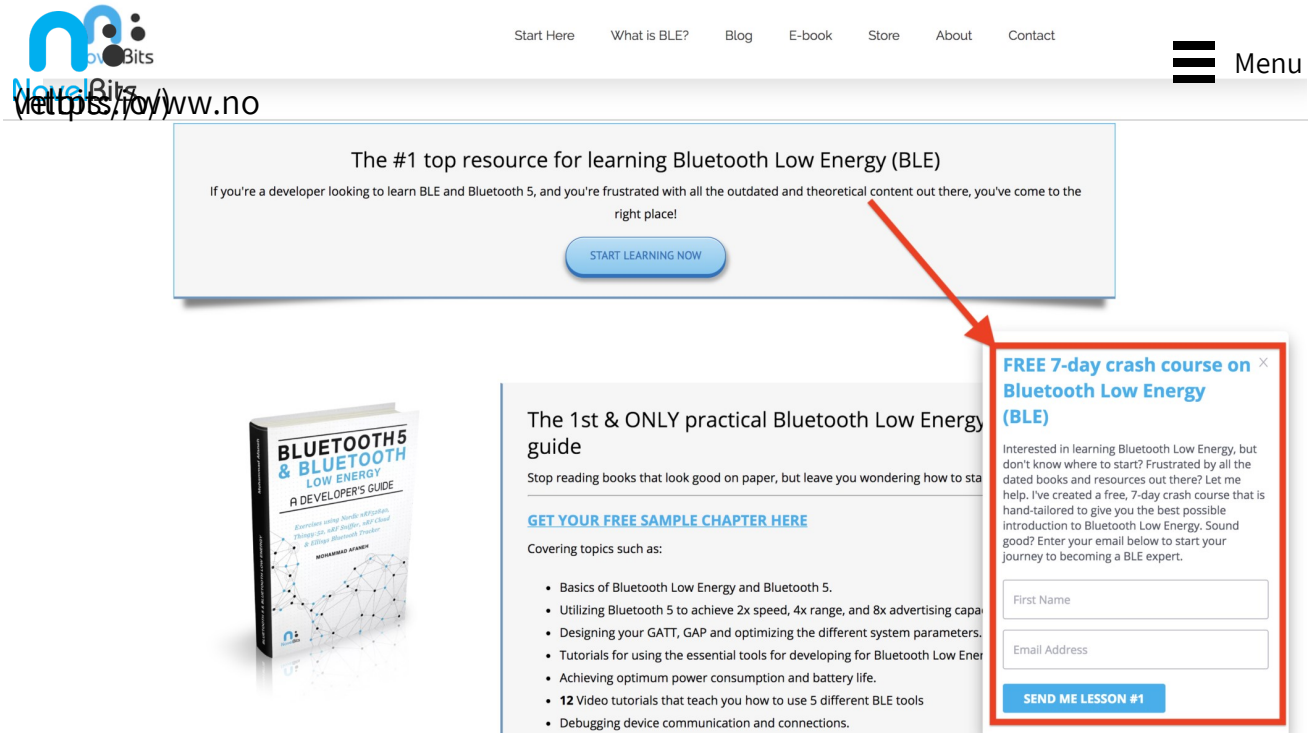
But if you start with a complex application, you can get lost, and ultimately become frustrated before you get something working.

…which is why we're keeping it simple in this application!

Prerequisites

To follow along with this example, you'll need to know the **basics of BLE**. It does not require you to have in-depth knowledge of BLE. In fact, I recommend you do not spend too much time going through the theory and skip right into developing a BLE application and getting your hands dirty once you've gone through learning the basics.

*If you're looking to learn the basics, or simply need a refresher, I recommend checking out my **FREE 7-day crash course on the Basics of Bluetooth Low Energy (BLE)**. Sign up by filling out the form in the bottom right-hand corner of the webpage:*



(https://www.novelbits.io/wp-content/uploads/2017/05/BLE-crash-course-pop-up-1.png)

Alternatively, you can sign up at the following link: **FREE 7-day crash course on Bluetooth Low Energy (BLE) (https://www.getdrip.com/forms/717220487/submissions/new).**

So, what hardware & software do I need?

For this tutorial, you'll need the following:

- **nRF52840 development kit** (although any nRF5x development will work, *with some modifications*)
- A PC running either Windows, Linux, or macOS (for development)
- Segger Embedded Studio (https://www.segger.com/downloads/embedded-studio/)

- nRF5 SDK (in our case, we're using the latest SDK provided by Nordic: nRF5 SDK (version 15.0.0)
- A mobile phone
- BLE client application running on the mobile phone (such as Nordic's nRF Connect or LightBlue)

Getting Started

Installing the SDK

We've already covered how to install Segger Embedded Studio (the FREE License IDE used for nRF5x development) in a previous post (https://www.novelbits.io/cross-platform-nrf-development-tutorial/). Here, we'll focus on the steps that follow the installation.

Next, let's download the latest nRF5 SDK (version 15.0.0) from Nordic's website (https://www.nordicsemi.com/eng/nordic/Products/nRF5-SDK/nRF5-SDK-zip/59011).

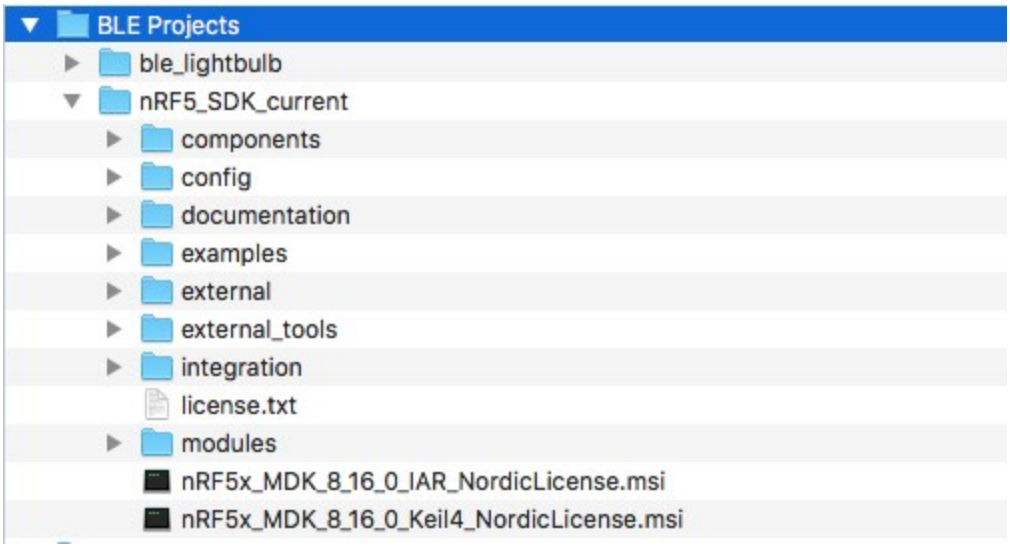
Once you have it downloaded, place it in a new folder. To make things easier, we'll put it in a folder alongside the application we'll be developing.

In my setup, I'll be creating a new folder named **"BLE Projects"** which contains the SDK folder as well as another folder for the application.

Folder Structure Setup

Create a new folder to hold the SDK as well as the application (*I named it "BLE Projects"*):

- Copy the SDK folder to the **"BLE Projects"** folder
- [Optional] Rename the SDK folder to **"nRF5_SDK_Current".** This way, if I need to migrate to a newer SDK, all I have to do is place it in that folder, and my current projects will not have to be updated (*other than any changes needed for the APIs to work with the newer SDK*)
- Navigate to the folder **examples/ble_peripheral/** and copy the folder **ble_app_template** to the **BLE Projects** folder
- Copying the example application folder outside of the SDK makes it easier to keep it self-contained and independent of any changes you make to the SDK folder (such as installing a newer SDK)
- Rename **ble_app_template** to another meaningful name that describes our simple application: **ble_lightbulb**



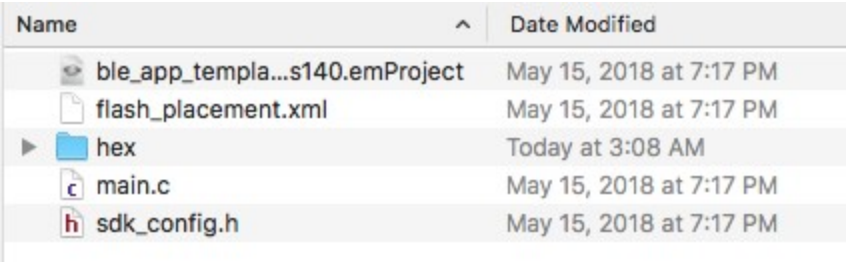
(https://www.novelbits.io/wp-content/uploads/2018/05/Screen-Shot-2018-05-29-at-3.00.00-AM.png)

Now, it's time for a little clean-up!

The example application folder contains a few unnecessary files and folders (related to other IDEs). I also find that it contains too many nested folders that make it harder to find important files. We'll do the following:

- Move the **sdk_config.h** file to the root folder of the application (located under **pca10056/s140/config**)
- Move the SES Solution file **(ble_app_template_pca10056_s140.emProject)** along with the **flash_placement.xml** file to the root application folder (both of these files are located under **pca10056/s140/ses**).
- You can also move the ***.jlink** and ***.emSession** files, but those are not necessary as SES will re-create them once you open the Solution.
- Delete the **ble_app_template.eww** file
- Delete the **pca*** folders (*don't forget to move the files listed above before you do this!)*

Here is what the resulting folder looks like for me:



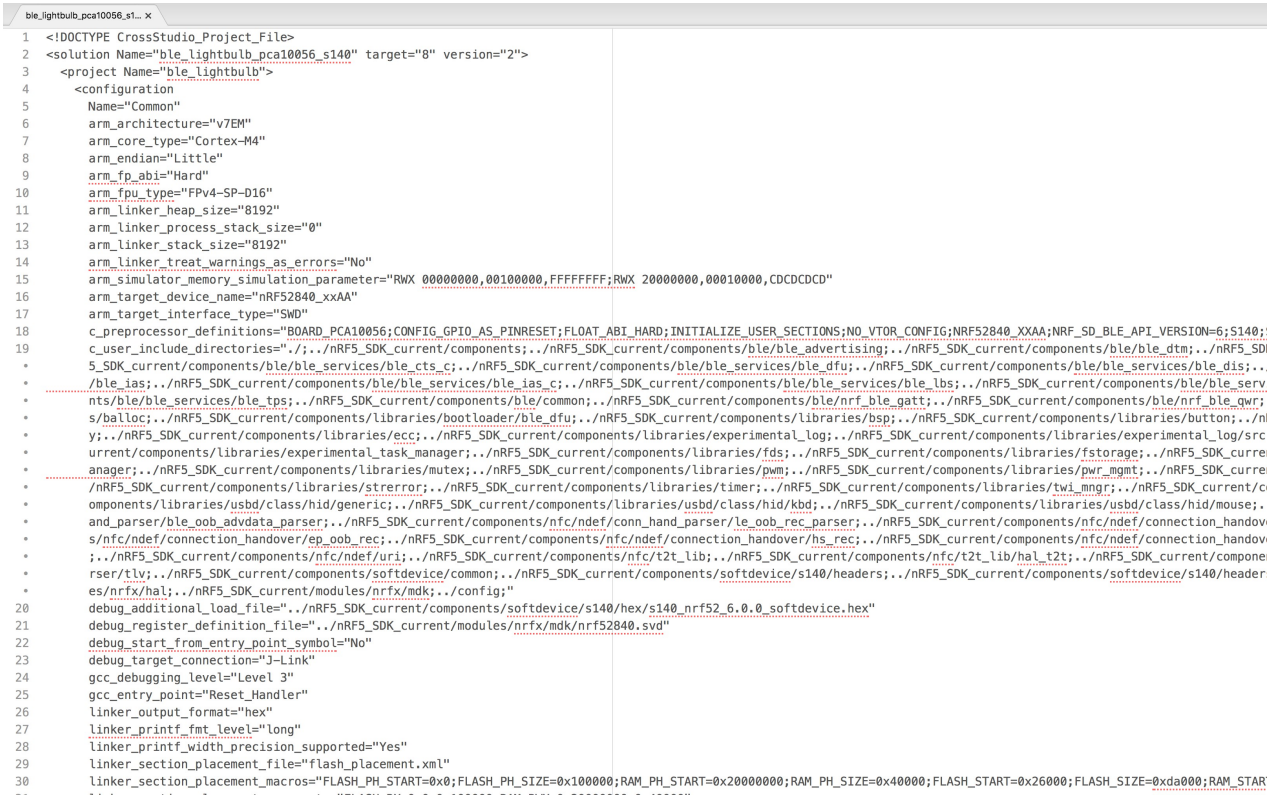
(https://www.novelbits.io/wp-content/uploads/2018/05/Screen-Shot-2018-05-29-at-3.08.55-AM.png)

SES Project File Configuration

Now that we've got the folder structure set up, we need to update the SES project file to reflect these changes (this is something we'll only have to do once).

First rename the file “**ble_app_template_pca10056_s140.emProject**” to “**ble_lightbulb_pca10056_s140.emProject**“.

Next, open the file in your text editor of choice.



(https://www.novelbits.io/wp-content/uploads/2018/05/Screen-Shot-2018-05-29-at-3.10.49-AM.png)

Here are the modifications we’re going to make to the project file (*I’ve already made all the changes and provided the full source code available for you to download if you want to skip these steps, at the end of this post*):

- Rename the Solution and Project names:

XHTML

```
<solution Name="ble_app_template_pca10056_s140" target="8" version="2">
  <project Name="ble_app_template_pca10056_s140">
```

```
XHTML
<solution Name="ble_lightbulb_pca10056_s140" target="8" version="2">
  <project Name="ble_lightbulb">
```

- Replace any references to the SDK (since we copied the application folder outside the SDK).

  Replace "../../../../.." with "nRF5_SDK_current"

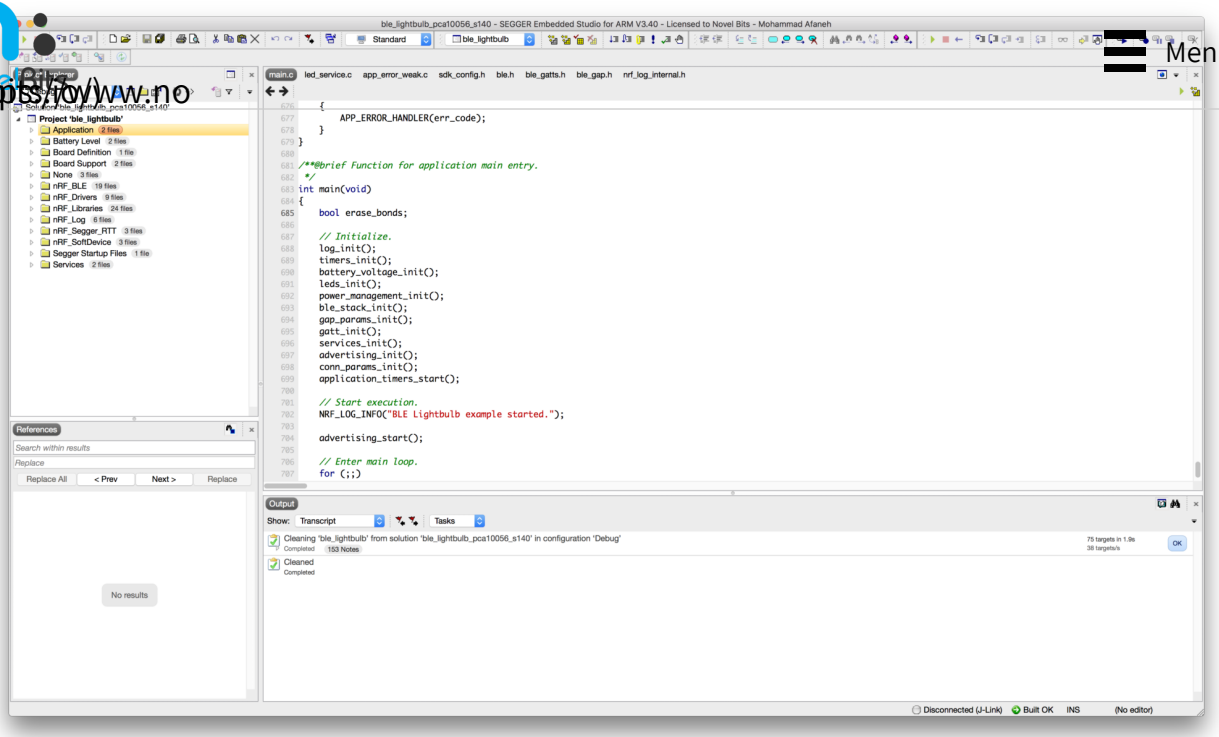
  E.g.:

```
XHTML
      <file file_name="../../../../../../components/libraries/experimental_log/
      <file file_name="../../../../../../components/libraries/experimental_log/
      <file file_name="../../../../../../components/libraries/experimental_log/
      <file file_name="../../../../../../components/libraries/experimental_log/
      <file file_name="../../../../../../components/libraries/experimental_log/
      <file file_name="../../../../../../components/libraries/experimental_log/
```

  replaced with:

```
XHTML
      <file file_name="../nRF5_SDK_current/components/libraries/experimental_lo
      <file file_name="../nRF5_SDK_current/components/libraries/experimental_lo
      <file file_name="../nRF5_SDK_current/components/libraries/experimental_lo
      <file file_name="../nRF5_SDK_current/components/libraries/experimental_lo
      <file file_name="../nRF5_SDK_current/components/libraries/experimental_lo
      <file file_name="../nRF5_SDK_current/components/libraries/experimental_lo
```

Next, open the Solution (*.emProject file) in SES.



(https://www.novelbits.io/wp-content/uploads/2018/05/Screen-Shot-2018-05-29-at-3.13.04-AM.png)

Now, make the following changes to the Project and the source files:

- In **main.c**, change the advertised name:

```c
#define DEVICE_NAME                     "BLE_Lightbulb"                         /
```

- Also in **main.c,** add the following lines (highlighted):

```c
```

```
#include "ble_bas.h"
#include "services/led_service.h"
#include "Battery Level/battery_voltage.h"


#define DEVICE_NAME                     "BLE_Lightbulb"
#define MANUFACTURER_NAME               "NordicSemiconductor"
#define APP_ADV_INTERVAL                300                                     /**<

// Corresponds to LED2 on the development kit
#define LIGHTBULB_LED                   BSP_BOARD_LED_1

#define APP_ADV_DURATION                18000
```

- Remove any peer manager references (only needed for security features):
  - Change

```c
static void advertising_start(bool erase_bonds);
```

to:

```c
static void advertising_start();
```

  - Remove the following block of code:

```c
```

```c
/**@brief Function for handling Peer Manager events.
 *
 * @param[in] p_evt  Peer Manager event.
 */
static void pm_evt_handler(pm_evt_t const * p_evt)
{
    ret_code_t err_code;

    switch (p_evt->evt_id)
    {
        case PM_EVT_BONDED_PEER_CONNECTED:
        {
            NRF_LOG_INFO("Connected to a previously bonded device.");
        } break;

        case PM_EVT_CONN_SEC_SUCCEEDED:
        {
            NRF_LOG_INFO("Connection secured: role: %d, conn_handle: 0
                         ble_conn_state_role(p_evt->conn_handle),
                         p_evt->conn_handle,
                         p_evt->params.conn_sec_succeeded.procedure);
        } break;

        case PM_EVT_CONN_SEC_FAILED:
        {
            /* Often, when securing fails, it shouldn't be restarted,
             * Other times, it can be restarted directly.
             * Sometimes it can be restarted, but only after changing
             * Sometimes, it cannot be restarted until the link is dis
             * Sometimes it is impossible, to secure the link, or the
             * How to handle this error is highly application dependen
        } break;

        case PM_EVT_CONN_SEC_CONFIG_REQ:
        {
            // Reject pairing request from an already bonded peer.
            pm_conn_sec_config_t conn_sec_config = {.allow_repairing =
            pm_conn_sec_config_reply(p_evt->conn_handle, &conn_sec_con
        } break;

        case PM_EVT_STORAGE_FULL:
        {
            // Run garbage collection on the flash.
            err_code = fds_gc();
            if (err_code == FDS_ERR_NO_SPACE_IN_QUEUES)
            {
                // Retry.
            }
            else
            {
                APP_ERROR_CHECK(err_code);
            }
        } break;

        case PM_EVT_PEERS_DELETE_SUCCEEDED:
        {
            advertising_start(false);
        } break;

        case PM_EVT_PEER_DATA_UPDATE_FAILED:
```

```
61          {
62              // Assert.
63              APP_ERROR_CHECK(p_evt->params.peer_data_update_failed.error);
64          } break;
65
66          case PM_EVT_PEER_DELETE_FAILED:
67          {
68              // Assert.
69              APP_ERROR_CHECK(p_evt->params.peer_delete_failed.error);
70          } break;
71
72          case PM_EVT_PEERS_DELETE_FAILED:
73          {
74              // Assert.
75              APP_ERROR_CHECK(p_evt->params.peers_delete_failed_evt.error);
76          } break;
77
78          case PM_EVT_ERROR_UNEXPECTED:
79          {
80              // Assert.
81              APP_ERROR_CHECK(p_evt->params.error_unexpected.error);
82          } break;
83
84          case PM_EVT_CONN_SEC_START:
85          case PM_EVT_PEER_DATA_UPDATE_SUCCEEDED:
86          case PM_EVT_PEER_DELETE_SUCCEEDED:
87          case PM_EVT_LOCAL_DB_CACHE_APPLIED:
88          case PM_EVT_LOCAL_DB_CACHE_APPLY_FAILED:
89              // This can happen when the local DB has changed.
90          case PM_EVT_SERVICE_CHANGED_IND_SENT:
91          case PM_EVT_SERVICE_CHANGED_IND_CONFIRMED:
92          default:
93              break;
94      }
95  }
```

◦ Delete the following block:

```
C
```

```c
/**@brief Function for the Peer Manager initialization.
 */
static void peer_manager_init(void)
{
    ble_gap_sec_params_t sec_param;
    ret_code_t           err_code;

    err_code = pm_init();
    APP_ERROR_CHECK(err_code);

    memset(&sec_param, 0, sizeof(ble_gap_sec_params_t));

    // Security parameters to be used for all security procedures.
    sec_param.bond           = SEC_PARAM_BOND;
    sec_param.mitm           = SEC_PARAM_MITM;
    sec_param.lesc           = SEC_PARAM_LESC;
    sec_param.keypress       = SEC_PARAM_KEYPRESS;
    sec_param.io_caps        = SEC_PARAM_IO_CAPABILITIES;
    sec_param.oob            = SEC_PARAM_OOB;
    sec_param.min_key_size   = SEC_PARAM_MIN_KEY_SIZE;
    sec_param.max_key_size   = SEC_PARAM_MAX_KEY_SIZE;
    sec_param.kdist_own.enc  = 1;
    sec_param.kdist_own.id   = 1;
    sec_param.kdist_peer.enc = 1;
    sec_param.kdist_peer.id  = 1;

    err_code = pm_sec_params_set(&sec_param);
    APP_ERROR_CHECK(err_code);

    err_code = pm_register(pm_evt_handler);
    APP_ERROR_CHECK(err_code);
}


/**@brief Clear bond information from persistent storage.
 */
static void delete_bonds(void)
{
    ret_code_t err_code;

    NRF_LOG_INFO("Erase bonds!");

    err_code = pm_peers_delete();
    APP_ERROR_CHECK(err_code);
}
```

◦ Change

```c
```

```c
/**@brief Function for starting advertising.
 */
static void advertising_start(bool erase_bonds)
{
    if (erase_bonds == true)
    {
        delete_bonds();
        // Advertising is started by PM_EVT_PEERS_DELETED_SUCEEDED event
    }
    else
    {
        ret_code_t err_code = ble_advertising_start(&m_advertising, BLE_

        APP_ERROR_CHECK(err_code);
    }
}
```

to

```c
/**@brief Function for starting advertising.
 */
static void advertising_start()
{
    ret_code_t err_code = ble_advertising_start(&m_advertising, BLE_ADV_M
    APP_ERROR_CHECK(err_code);
}
```

- Remove the highlighted line:

```c
    services_init();
    conn_params_init();
    peer_manager_init();
```

- Change

```c
    advertising_start(erase_bonds);
```

to

```c
    advertising_init();
```

- Modify the following (w*e are not using any buttons in our application, so we only need to initialize the LEDs):*

```c
/**@brief Function for initializing buttons and leds.
 *
 * @param[out] p_erase_bonds  Will be true if the clear bonding button w
 */
static void buttons_leds_init(bool * p_erase_bonds)
{
    ret_code_t err_code;
    bsp_event_t startup_event;

    err_code = bsp_init(BSP_INIT_LEDS | BSP_INIT_BUTTONS, bsp_event_hand
    APP_ERROR_CHECK(err_code);

    err_code = bsp_btn_ble_init(NULL, &startup_event);
    APP_ERROR_CHECK(err_code);

    *p_erase_bonds = (startup_event == BSP_EVENT_CLEAR_BONDING_DATA);
}
```

to

```c
 *
 * @details Initializes all LEDs used by the application.
 */
static void leds_init()
{
    ret_code_t err_code;

    err_code = bsp_init(BSP_INIT_LEDS, bsp_event_handler);
    APP_ERROR_CHECK(err_code);
}
```

- In main(), modify

```c
buttons_leds_init(&erase_bonds);
```

(https://www.no

to

```c
leds_init();
```

- Change the debug print out for the main application:

```c
NRF_LOG_INFO("Template example started.");
```

to

```c
NRF_LOG_INFO("BLE Lightbulb example started.");
```

- We need to add a write handler for the LED settings in **main.c**. This function will get used for the custom service and characteristic we'll be implementing in our application.

```c
/**@brief Function for handling write events to the LED characteristic.
 *
 * @param[in] p_led_service  Instance of LED Service to which the write applie
 * @param[in] led_state      Written/desired state of the LED.
 */
static void led_write_handler(uint16_t conn_handle, ble_led_service_t * p_led_
{
    if (led_state)
    {
        bsp_board_led_on(LIGHTBULB_LED);
        NRF_LOG_INFO("Received LED ON!");
    }
    else
    {
        bsp_board_led_off(LIGHTBULB_LED);
        NRF_LOG_INFO("Received LED OFF!");
    }
}
```

Let's talk about the design of our application, and what elements we need to add to implement our smart BLE-controlled lightbulb.

The application allows the user to:

- Discover the BLE Peripheral named "**BLE_Lightbulb**" from a BLE Central application
- Connect to the Peripheral and discover the Services and Characteristics exposed by the Peripheral
- Turn ON/OFF LED2 on the nRF52840 development kit as well as read the status of the LED (whether ON or OFF)
- Subscribe to Notifications to the Battery Level Characteristic exposed by the Peripheral inside the **Battery Service**

LED Service + LED 2 Characteristic

To be able to control the LED on the development kit from a BLE client (a smartphone application in our case), we'll need to implement a custom service and characteristic for the LED.

iIn a previous post (Custom Services and Characteristics [BLE MIDI use case](https://www.novelbits.io/bluetooth-gatt-services-characteristics/)), we described how to create custom services and characteristics, but here's all you need to know:

- We need to create one service. We'll call it **led_service**.
- In that service, we'll add a single characteristic dedicated to LED 2 on our development kit (LED 1 will be used to indicate the state of the BLE peripheral: Advertising or Connected). Let's call this characteristic: **led_2_char**.

- For each (service and characteristic), we'll need to choose a custom UUID (to learn more about custom/vendor-specific UUIDs and how to pick them, refer to this blog post (Creating custom UUIDs (https://www.novelbits.io/uuid-for-custom-services-and-characteristics/)).
- We'll define the UUIDs as following:
    - LED Service UUID: **E54B0001-67F5-479E-8711-B3B99198CE6C**
    - LED 2 Characteristic UUID: **E54B0002-67F5-479E-8711-B3B99198CE6C**
- The LED 2 Characteristic will have the following permissions:
    - Write-enabled
    - Read-enabled
    - NO Notifications or Indications enabled (since we'll be controlling the LED from the BLE client)

Implementation

Instead of including the implementation of the LED service in the **main.c** file, we'll be including them in a separate folder and files.

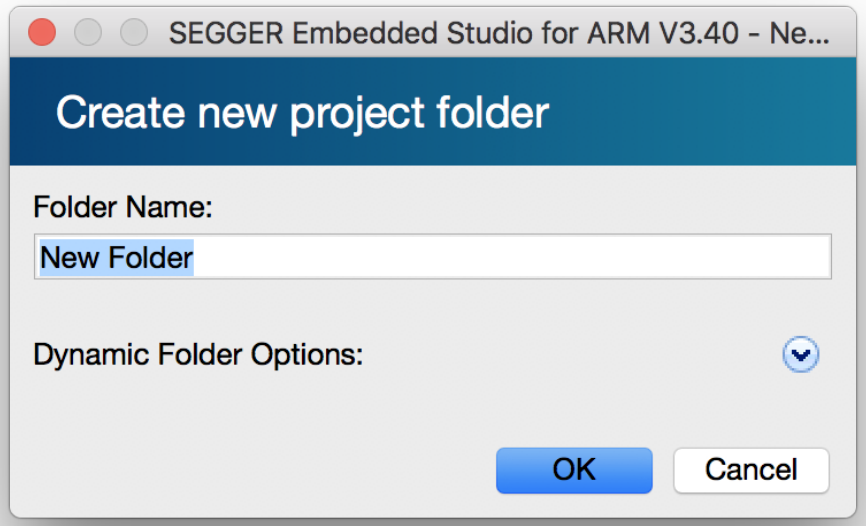
Let's create two files: **led_service.h** & **led_service.c.**

We'll be placing these in a separate folder named **Services**.

- Create a folder named **Services** in the root folder of the application (from your Operating System's file explorer)
- Right-click on **Project 'ble_lightbulb'**

- Click **New Folder**

(https://www.novelbits.io/wp-content/uploads/2018/05/Screen-Shot-2018-05-29-at-3.21.40-AM.png)

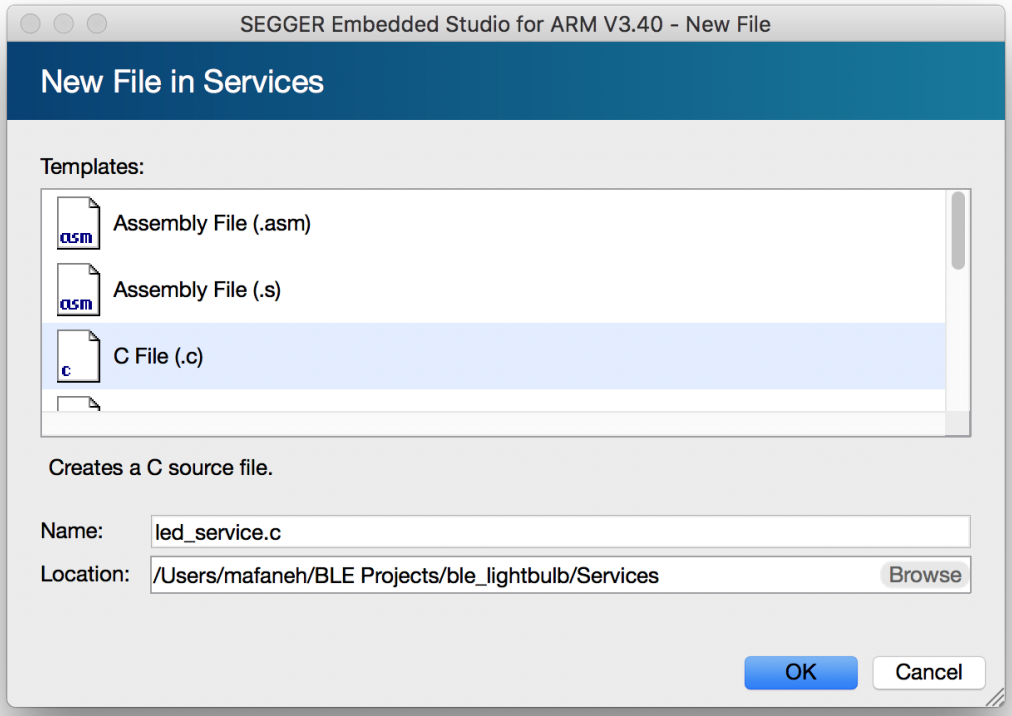


(https://www.novelbits.io/wp-content/uploads/2018/05/Screen-Shot-2018-05-29-at-3.21.40-AM.png)

- Rename **New Folder** to **Services**
- Now you should have an empty **Services** folder
- Right-click on **Services** and choose to **Add New File**
- Choose **C File (.c)**, and type **led_service.c** in the **Name** field
- Under **Location**, click **Browse** and navigate to the **Services** folder

- Select **Services** and click **Choose**

(https://www.novelbits.io/wp-content/uploads/2018/05/Screen-Shot-2018-05-29-at-3.31.30-AM.png)



(https://www.novelbits.io/wp-content/uploads/2018/05/Screen-Shot-2018-05-29-at-3.31.30-AM.png)

- Click **OK**
- Repeat the steps above but choose **Header File (.h)** instead and enter **led_service.h** in the **Name** field

Here are the contents of the source files. We'll list the full source code first, and then go through the content explaining it.

led_service.h

```
C
```

```
/*
 * The MIT License (MIT)
 * Copyright (c) 2018 Novel Bits
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * to deal in the Software without restriction, including without limitation the
 * and/or sell copies of the Software, and to permit persons to whom the Softwar
 *
 * The above copyright notice and this permission notice shall be included in al
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IM
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE A
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * IN THE SOFTWARE.
 *
 */

#ifndef LED_SERVICE_H
#define LED_SERVICE_H

#include
#include "boards.h"
#include "ble.h"
#include "ble_srv_common.h"
#include "nrf_sdh_ble.h"

/**@brief   Macro for defining a ble_led_service instance.
 *
 * @param   _name   Name of the instance.
 * @hideinitializer
 */

#define BLE_LED_SERVICE_BLE_OBSERVER_PRIO 2

#define BLE_LED_SERVICE_DEF(_name)
static ble_led_service_t _name;
NRF_SDH_BLE_OBSERVER(_name ## _obs,
                     BLE_LED_SERVICE_BLE_OBSERVER_PRIO,
                     ble_led_service_on_ble_evt, &amp;_name)

// LED service:                     E54B0001-67F5-479E-8711-B3B99198CE6C
//   LED 2 characteristic:   E54B0002-67F5-479E-8711-B3B99198CE6C

// The bytes are stored in little-endian format, meaning the
// Least Significant Byte is stored first
// (reversed from the order they're displayed as)

// Base UUID: E54B0000-67F5-479E-8711-B3B99198CE6C
#define BLE_UUID_LED_SERVICE_BASE_UUID  {0x6C, 0xCE, 0x98, 0x91, 0xB9, 0xB3, 0x1

// Service &amp;amp; characteristics UUIDs
#define BLE_UUID_LED_SERVICE_UUID  0x0001
#define BLE_UUID_LED_2_CHAR_UUID   0x0002

// Forward declaration of the custom_service_t type.
typedef struct ble_led_service_s ble_led_service_t;

typedef void (*ble_led_service_led_write_handler_t) (uint16_t conn_handle, ble_l

/** @brief LED Service init structure. This structure contains all options and d
```

```
 *        initialization of the service.*/
typedef struct
{
    ble_led_service_led_write_handler_t led_write_handler; /**&lt; Event handler t
} ble_led_service_init_t;

/**@brief LED Service structure.
 *        This contains various status information
 *        for the service.
 */
typedef struct ble_led_service_s
{
    uint16_t                           conn_handle;
    uint16_t                           service_handle;
    uint8_t                            uuid_type;
    ble_gatts_char_handles_t           led_2_char_handles;
    ble_led_service_led_write_handler_t led_write_handler;

} ble_led_service_t;

// Function Declarations

/**@brief Function for initializing the LED Service.
 *
 * @param[out]  p_led_service  LED Service structure. This structure will have to
 *                             the application. It will be initialized by this
 *                             be used to identify this particular service inst
 *
 * @return      NRF_SUCCESS on successful initialization of service, otherwise an
 */
uint32_t ble_led_service_init(ble_led_service_t * p_led_service, const ble_led_ser

/**@brief Function for handling the application's BLE stack events.
 *
 * @details This function handles all events from the BLE stack that are of intere
 *
 * @param[in] p_ble_evt  Event received from the BLE stack.
 * @param[in] p_context  LED Service structure.
 */
void ble_led_service_on_ble_evt(ble_evt_t const * p_ble_evt, void * p_context);

#endif /* LED_SERVICE_H */
```

led_service.c

```c
```

```c
/*
 * The MIT License (MIT)
 * Copyright (c) 2018 Novel Bits
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * to deal in the Software without restriction, including without limitation the
 * and/or sell copies of the Software, and to permit persons to whom the Softwar
 *
 * The above copyright notice and this permission notice shall be included in al
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IM
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE A
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * IN THE SOFTWARE.
 *
 */

#include

#include "nrf_log.h"
#include "led_service.h"

static const uint8_t LED2CharName[] = "LED 2";

/**@brief Function for handling the Connect event.
 *
 * @param[in]   p_led_service  LED service structure.
 * @param[in]   p_ble_evt      Event received from the BLE stack.
 */
static void on_connect(ble_led_service_t * p_led_service, ble_evt_t const * p_bl
{
    p_led_service-&gt;conn_handle = p_ble_evt-&gt;evt.gap_evt.conn_handle;
}

/**@brief Function for handling the Disconnect event.
 *
 * @param[in]   p_bas       LED service structure.
 * @param[in]   p_ble_evt   Event received from the BLE stack.
 */
static void on_disconnect(ble_led_service_t * p_led_service, ble_evt_t const * p
{
    UNUSED_PARAMETER(p_ble_evt);
    p_led_service-&gt;conn_handle = BLE_CONN_HANDLE_INVALID;
}

/**@brief Function for handling the Write event.
 *
 * @param[in] p_led_service   LED Service structure.
 * @param[in] p_ble_evt       Event received from the BLE stack.
 */
static void on_write(ble_led_service_t * p_led_service, ble_evt_t const * p_ble_
{
    ble_gatts_evt_write_t const * p_evt_write = &amp;p_ble_evt-&gt;evt.gatts_evt

    if (   (p_evt_write-&gt;handle == p_led_service-&gt;led_2_char_handles.value
        &amp;&amp; (p_evt_write-&gt;len == 1)
        &amp;&amp; (p_led_service-&gt;led_write_handler != NULL))
    {
        p_led_service-&gt;led_write_handler(p_ble_evt-&gt;evt.gap_evt.conn_handl
```

```
}

/**@brief Function for adding the LED 2 characteristic.
 */
static uint32_t led_2_char_add(ble_led_service_t * p_led_service)
{
    ble_gatts_char_md_t char_md;
    ble_gatts_attr_t    attr_char_value;
    ble_gatts_attr_md_t attr_md;
    ble_uuid_t          ble_uuid;

    memset(&amp;char_md, 0, sizeof(char_md));
    memset(&amp;attr_md, 0, sizeof(attr_md));
    memset(&amp;attr_char_value, 0, sizeof(attr_char_value));

    char_md.char_props.read          = 1;
    char_md.char_props.write         = 1;
    char_md.p_char_user_desc         = LED2CharName;
    char_md.char_user_desc_size      = sizeof(LED2CharName);
    char_md.char_user_desc_max_size  = sizeof(LED2CharName);
    char_md.p_char_pf                = NULL;
    char_md.p_user_desc_md           = NULL;
    char_md.p_cccd_md                = NULL;
    char_md.p_sccd_md                = NULL;

    // Define the LED 2 Characteristic UUID
    ble_uuid.type = p_led_service-&gt;uuid_type;
    ble_uuid.uuid = BLE_UUID_LED_2_CHAR_UUID;

    // Set permissions on the Characteristic value
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&amp;attr_md.write_perm);
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&amp;attr_md.read_perm);

    // Attribute Metadata settings
    attr_md.vloc       = BLE_GATTS_VLOC_STACK;
    attr_md.rd_auth    = 0;
    attr_md.wr_auth    = 0;
    attr_md.vlen       = 0;

    // Attribute Value settings
    attr_char_value.p_uuid       = &amp;ble_uuid;
    attr_char_value.p_attr_md    = &amp;attr_md;
    attr_char_value.init_len     = sizeof(uint8_t);
    attr_char_value.init_offs    = 0;
    attr_char_value.max_len      = sizeof(uint8_t);
    attr_char_value.p_value      = NULL;

    return sd_ble_gatts_characteristic_add(p_led_service-&gt;service_handle, &am
                                           &amp;attr_char_value,
                                           &amp;p_led_service-&gt;led_2_char_han
}

uint32_t ble_led_service_init(ble_led_service_t * p_led_service, const ble_led_s
{
    uint32_t   err_code;
    ble_uuid_t ble_uuid;

    // Initialize service structure
    p_led_service-&gt;conn_handle = BLE_CONN_HANDLE_INVALID;
```

```
121
122     // Initialize service structure.
123     p_led_service-&gt;led_write_handler = p_led_service_init-&gt;led_write_handler
124
125     // Add service UUID
126     ble_uuid128_t base_uuid = {BLE_UUID_LED_SERVICE_BASE_UUID};
127     err_code = sd_ble_uuid_vs_add(&amp;base_uuid, &amp;p_led_service-&gt;uuid_type
128     if (err_code != NRF_SUCCESS)
129     {
130         return err_code;
131     }
132
133     // Set up the UUID for the service (base + service-specific)
134     ble_uuid.type = p_led_service-&gt;uuid_type;
135     ble_uuid.uuid = BLE_UUID_LED_SERVICE_UUID;
136
137     // Set up and add the service
138     err_code = sd_ble_gatts_service_add(BLE_GATTS_SRVC_TYPE_PRIMARY, &amp;ble_uuid
139     if (err_code != NRF_SUCCESS)
140     {
141         return err_code;
142     }
143
144     // Add the different characteristics in the service:
145     //   Button press characteristic:   E54B0002-67F5-479E-8711-B3B99198CE6C
146     err_code = led_2_char_add(p_led_service);
147     if (err_code != NRF_SUCCESS)
148     {
149         return err_code;
150     }
151
152     return NRF_SUCCESS;
153 }
154
155 void ble_led_service_on_ble_evt(ble_evt_t const * p_ble_evt, void * p_context)
156 {
157     ble_led_service_t * p_led_service = (ble_led_service_t *)p_context;
158
159     switch (p_ble_evt-&gt;header.evt_id)
160     {
161         case BLE_GAP_EVT_CONNECTED:
162             on_connect(p_led_service, p_ble_evt);
163             break;
164
165         case BLE_GATTS_EVT_WRITE:
166             on_write(p_led_service, p_ble_evt);
167             break;
168
169         case BLE_GAP_EVT_DISCONNECTED:
170             on_disconnect(p_led_service, p_ble_evt);
171             break;
172
173         default:
174             // No implementation needed.
175             break;
176     }
177 }
```

Let’s explain the most important elements of the implementation:

led_service.h

- Lines 33-39: define a macro that will be used in **main.c** to instantiate the LED service data structure

```c
#define BLE_LED_SERVICE_BLE_OBSERVER_PRIO 2

#define BLE_LED_SERVICE_DEF(_name)
static ble_led_service_t _name;
NRF_SDH_BLE_OBSERVER(_name ## _obs,
                     BLE_LED_SERVICE_BLE_OBSERVER_PRIO,
                     ble_led_service_on_ble_evt, &_name)
```

- Lines 41-53: define the UUIDs for the LED Service and the LED 2 Characteristic

```c
// LED service:                    E54B0001-67F5-479E-8711-B3B99198CE6C
//   LED 2 characteristic:   E54B0002-67F5-479E-8711-B3B99198CE6C

// The bytes are stored in little-endian format, meaning the
// Least Significant Byte is stored first
// (reversed from the order they're displayed as)

// Base UUID: E54B0000-67F5-479E-8711-B3B99198CE6C
#define BLE_UUID_LED_SERVICE_BASE_UUID  {0x6C, 0xCE, 0x98, 0x91, 0xB9, 0xB3, 0

// Service & characteristics UUIDs
#define BLE_UUID_LED_SERVICE_UUID  0x0001
#define BLE_UUID_LED_2_CHAR_UUID   0x0002
```

- Lines 55-65: define the led write handler function prototype and the LED service initialization data structure

```c
```

```
// Forward declaration of the custom_service_t type.
typedef struct ble_led_service_s ble_led_service_t;

typedef void (*ble_led_service_led_write_handler_t) (uint16_t conn_handle, ble

/** @brief LED Service init structure. This structure contains all options and
 *        initialization of the service.*/
typedef struct
{
    ble_led_service_led_write_handler_t led_write_handler; /**< Event handler
} ble_led_service_init_t;
```

- Lines 67-79: define the main LED service data structure that stores all the information relevant to the service

```c
/**@brief LED Service structure.
 *        This contains various status information
 *        for the service.
 */
typedef struct ble_led_service_s
{
    uint16_t                          conn_handle;
    uint16_t                          service_handle;
    uint8_t                           uuid_type;
    ble_gatts_char_handles_t          led_2_char_handles;
    ble_led_service_led_write_handler_t led_write_handler;

} ble_led_service_t;
```

- Lines 81-100: declare the functions needed for initializing the service as well as the event handler

```c
```

```c
// Function Declarations

/**@brief Function for initializing the LED Service.
 *
 * @param[out]  p_led_service  LED Service structure. This structure will have
 *                             the application. It will be initialized by t
 *                             be used to identify this particular service
 *
 * @return      NRF_SUCCESS on successful initialization of service, otherwise
 */
uint32_t ble_led_service_init(ble_led_service_t * p_led_service, const ble_led

/**@brief Function for handling the application's BLE stack events.
 *
 * @details This function handles all events from the BLE stack that are of ir
 *
 * @param[in] p_ble_evt  Event received from the BLE stack.
 * @param[in] p_context  LED Service structure.
 */
void ble_led_service_on_ble_evt(ble_evt_t const * p_ble_evt, void * p_context)
```

led_service.c

- Lines 18-23: include the necessary header files as well as define the string

```c
#include <string.h>

#include "nrf_log.h"
#include "led_service.h"

static const uint8_t LED2CharName[] = "LED 2";
```

- Lines 25-44: define the functions for handling the connection and disconnection events

```c
```

```
/**@brief Function for handling the Connect event.
 *
 * @param[in]   p_led_service  LED service structure.
 * @param[in]   p_ble_evt      Event received from the BLE stack.
 */
static void on_connect(ble_led_service_t * p_led_service, ble_evt_t const * p_
{
    p_led_service->conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
}

/**@brief Function for handling the Disconnect event.
 *
 * @param[in]   p_bas       LED service structure.
 * @param[in]   p_ble_evt   Event received from the BLE stack.
 */
static void on_disconnect(ble_led_service_t * p_led_service, ble_evt_t const *
{
    UNUSED_PARAMETER(p_ble_evt);
    p_led_service->conn_handle = BLE_CONN_HANDLE_INVALID;
}
```

- Lines 46-61: define the function for handling the BLE write event

```
/**@brief Function for handling the Write event.
 *
 * @param[in] p_led_service   LED Service structure.
 * @param[in] p_ble_evt       Event received from the BLE stack.
 */
static void on_write(ble_led_service_t * p_led_service, ble_evt_t const * p_bl
{
    ble_gatts_evt_write_t const * p_evt_write = &p_ble_evt->evt.gatts_evt.para

    if (   (p_evt_write->handle == p_led_service->led_2_char_handles.value_har
        && (p_evt_write->len == 1)
        && (p_led_service->led_write_handler != NULL))
    {
        p_led_service->led_write_handler(p_ble_evt->evt.gap_evt.conn_handle, p
    }
}
```

- Lines 63-112: define the function for adding the LED 2 Characteristic

```

```

```c
/**@brief Function for adding the LED 2 characteristic.
 *
 */
static uint32_t led_2_char_add(ble_led_service_t * p_led_service)
{
    ble_gatts_char_md_t char_md;
    ble_gatts_attr_t    attr_char_value;
    ble_gatts_attr_md_t attr_md;
    ble_uuid_t          ble_uuid;

    memset(&char_md, 0, sizeof(char_md));
    memset(&attr_md, 0, sizeof(attr_md));
    memset(&attr_char_value, 0, sizeof(attr_char_value));

    char_md.char_props.read          = 1;
    char_md.char_props.write         = 1;
    char_md.p_char_user_desc         = LED2CharName;
    char_md.char_user_desc_size      = sizeof(LED2CharName);
    char_md.char_user_desc_max_size  = sizeof(LED2CharName);
    char_md.p_char_pf                = NULL;
    char_md.p_user_desc_md           = NULL;
    char_md.p_cccd_md                = NULL;
    char_md.p_sccd_md                = NULL;

    // Define the LED 2 Characteristic UUID
    ble_uuid.type = p_led_service->uuid_type;
    ble_uuid.uuid = BLE_UUID_LED_2_CHAR_UUID;

    // Set permissions on the Characteristic value
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&attr_md.write_perm);
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&attr_md.read_perm);

    // Attribute Metadata settings
    attr_md.vloc       = BLE_GATTS_VLOC_STACK;
    attr_md.rd_auth    = 0;
    attr_md.wr_auth    = 0;
    attr_md.vlen       = 0;

    // Attribute Value settings
    attr_char_value.p_uuid       = &ble_uuid;
    attr_char_value.p_attr_md    = &attr_md;
    attr_char_value.init_len     = sizeof(uint8_t);
    attr_char_value.init_offs    = 0;
    attr_char_value.max_len      = sizeof(uint8_t);
    attr_char_value.p_value      = NULL;

    return sd_ble_gatts_characteristic_add(p_led_service->service_handle, &cha
                                           &attr_char_value,
                                           &p_led_service->led_2_char_handles)
}
```

- Lines 114-153: define the function used to initialize the LED Service

```c
```

```c
uint32_t ble_led_service_init(ble_led_service_t * p_led_service, const ble_led
{
    uint32_t   err_code;
    ble_uuid_t ble_uuid;

    // Initialize service structure
    p_led_service->conn_handle = BLE_CONN_HANDLE_INVALID;

    // Initialize service structure.
    p_led_service->led_write_handler = p_led_service_init->led_write_handler;

    // Add service UUID
    ble_uuid128_t base_uuid = {BLE_UUID_LED_SERVICE_BASE_UUID};
    err_code = sd_ble_uuid_vs_add(&base_uuid, &p_led_service->uuid_type);
    if (err_code != NRF_SUCCESS)
    {
        return err_code;
    }

    // Set up the UUID for the service (base + service-specific)
    ble_uuid.type = p_led_service->uuid_type;
    ble_uuid.uuid = BLE_UUID_LED_SERVICE_UUID;

    // Set up and add the service
    err_code = sd_ble_gatts_service_add(BLE_GATTS_SRVC_TYPE_PRIMARY, &ble_uuid
    if (err_code != NRF_SUCCESS)
    {
        return err_code;
    }

    // Add the different characteristics in the service:
    //   Button press characteristic:   E54B0002-67F5-479E-8711-B3B99198CE6C
    err_code = led_2_char_add(p_led_service);
    if (err_code != NRF_SUCCESS)
    {
        return err_code;
    }

    return NRF_SUCCESS;
}
```

- Lines 155-177: define the BLE event handler that gets called by the SoftDevice (to handle the different events such as connection, disconnection, and write events)

```c
```

```
void ble_led_service_on_ble_evt(ble_evt_t const * p_ble_evt, void * p_context)
{
    ble_led_service_t * p_led_service = (ble_led_service_t *)p_context;

    switch (p_ble_evt-&gt;header.evt_id)
    {
        case BLE_GAP_EVT_CONNECTED:
            on_connect(p_led_service, p_ble_evt);
            break;

        case BLE_GATTS_EVT_WRITE:
            on_write(p_led_service, p_ble_evt);
            break;

        case BLE_GAP_EVT_DISCONNECTED:
            on_disconnect(p_led_service, p_ble_evt);
            break;

        default:
            // No implementation needed.
            break;
    }
}
```

Battery Service

In addition to the custom Service and Characteristic, we'll be using the Bluetooth SIG defined **Battery Service**, which includes a **Battery Level Characteristic**. (No need to create our own custom since we can reuse the SIG-defined Service).

Nordic's SDK already has this Service implemented, so all we need to do is:

- Enable the Battery Service in **sdk_config.h**. You can do so by setting the following macro to **1**:

```c
// <e> BLE_BAS_ENABLED - ble_bas - Battery Service
//==========================================================
#ifndef BLE_BAS_ENABLED
#define BLE_BAS_ENABLED 1
#endif
```

- Add the **ble_bas.c** file:

    - Right-click on the folder named **nRF_BLE** under the Project in the Project Explorer window
    - Click on **Add Existing File**
    - Locate the **ble_bas.c** file under **<SDK folder>/components/ble/ble_services/ble_bas**
    - Click on **ble_bas.c** and hit **Add**
- Update **NRF_SDH_BLE_VS_UUID_COUNT** in **sdk_config.h** to reflect the number of custom UUIDs (*2 in our case*: one for the LED Service and one for the LED 2 Characteristic)

```c
// <o> NRF_SDH_BLE_VS_UUID_COUNT - The number of vendor-specific UUIDs.
#ifndef NRF_SDH_BLE_VS_UUID_COUNT
#define NRF_SDH_BLE_VS_UUID_COUNT 2
#endif
```

Now that we have the Battery Service enabled, we need to implement the actual reading of the battery voltage level from the coin-cell battery installed in the development kit.

The good thing is we don't have to implement this from scratch. In fact, one of the components within the nRF5 SDK provides exactly what we need!

The component we'll be borrowing the code from is the **Eddystone component (used for a beacon standard called Eddystone)**.

Navigate to the folder **<SDK folder>/components/ble/ble_services/eddystone**

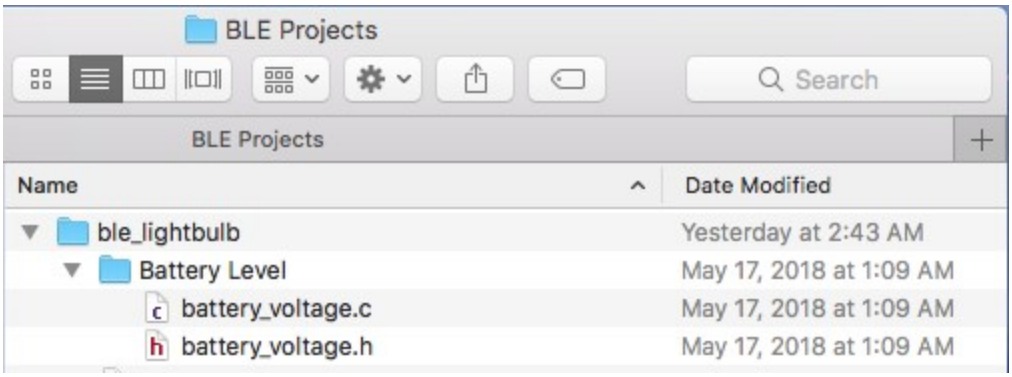
There you will find the files:

es_battery_voltage.h

es_battery_voltage_saadc.c

We'll copy these files into our project folder "**ble_lightbulb**" under a newly created folder named **Battery Level**.

Then rename them to "**battery_voltage.c**" and "**battery_voltage.h**":



(https://www.novelbits.io/wp-content/uploads/2018/05/Screen-Shot-2018-05-28-at-1.47.55-PM.png)

Now, you'll need to add this folder to the Project in SES. Follow the same steps we did before with the LED Service files (creating a folder named **Battery Level**, but then using the **Add Existing File** option instead of creating a new one).

We'll make a few slight modifications to functions names and such:

battery_voltage.h

```c
#ifndef BATTERY_VOLTAGE_H__
#define BATTERY_VOLTAGE_H__

#include

/**@brief Function for initializing the battery voltage module.
 */
void battery_voltage_init(void);

/**@brief Function for reading the battery voltage.
 *
 * @param[out]   p_vbatt       Pointer to the battery voltage value.
 */
void battery_voltage_get(uint16_t * p_vbatt);

#endif // BATTERY_VOLTAGE_H__
```

battery_voltage.c

(https://www.no

C

```
#include "battery_voltage.h"
#include "nrf_drv_saadc.h"
#include "sdk_macros.h"
#include "nrf_log.h"

#define ADC_REF_VOLTAGE_IN_MILLIVOLTS  600  //!&lt; Reference voltage (in milli v
#define DIODE_FWD_VOLT_DROP_MILLIVOLTS 270  //!&lt; Typical forward voltage drop
#define ADC_RES_10BIT                  1024 //!&lt; Maximum digital value for 10-
#define ADC_PRE_SCALING_COMPENSATION   6    //!&lt; The ADC is configured to use
#define ADC_RESULT_IN_MILLI_VOLTS(ADC_VALUE) \
    ((((ADC_VALUE) *ADC_REF_VOLTAGE_IN_MILLIVOLTS) / ADC_RES_10BIT) * ADC_PRE_SCA

static nrf_saadc_value_t adc_buf;                   //!&lt; Buffer used for stori
static uint16_t          m_batt_lvl_in_milli_volts; //!&lt; Current battery level

/**@brief Function handling events from 'nrf_drv_saadc.c'.
 *
 * @param[in] p_evt SAADC event.
 */
static void saadc_event_handler(nrf_drv_saadc_evt_t const * p_evt)
{
    if (p_evt-&gt;type == NRF_DRV_SAADC_EVT_DONE)
    {
        nrf_saadc_value_t adc_result;

        adc_result = p_evt-&gt;data.done.p_buffer[0];

        m_batt_lvl_in_milli_volts = ADC_RESULT_IN_MILLI_VOLTS(adc_result) + DIODE

        NRF_LOG_INFO("ADC reading - ADC:%d,  In Millivolts: %d\r\n", adc_result,
    }
}

void battery_voltage_init(void)
{
    ret_code_t err_code = nrf_drv_saadc_init(NULL, saadc_event_handler);

    APP_ERROR_CHECK(err_code);

    nrf_saadc_channel_config_t config =
        NRF_DRV_SAADC_DEFAULT_CHANNEL_CONFIG_SE(NRF_SAADC_INPUT_VDD);
    err_code = nrf_drv_saadc_channel_init(0, &amp;config);
    APP_ERROR_CHECK(err_code);

    err_code = nrf_drv_saadc_buffer_convert(&amp;adc_buf, 1);
    APP_ERROR_CHECK(err_code);

    err_code = nrf_drv_saadc_sample();
    APP_ERROR_CHECK(err_code);
}

void battery_voltage_get(uint16_t * p_vbatt)
{
    VERIFY_PARAM_NOT_NULL_VOID(p_vbatt);

    *p_vbatt = m_batt_lvl_in_milli_volts;
    if (!nrf_drv_saadc_is_busy())
    {
        ret_code_t err_code = nrf_drv_saadc_buffer_convert(&amp;adc_buf, 1);
```

```
    err_code = nrf_drv_saadc_sample();
    APP_ERROR_CHECK(err_code);
}
```

To be able to read the battery level, we have to set up a timer to trigger the reading of the battery voltage in the application's **main.c** file.

Here are the changes we need to make to implement this functionality:

- Define the battery timer and the battery level update period:

```c
/**< Battery timer. */
APP_TIMER_DEF(m_battery_timer_id);

#define BATTERY_LEVEL_MEAS_INTERVAL     APP_TIMER_TICKS(120000)
```

- Declare two functions for handling the timeout and another for updating the battery level:

```c
/**@brief Function for handling the Battery measurement timer timeout.
 *
 * @details This function will be called each time the battery level measureme
 *
 * @param[in] p_context  Pointer used for passing some arbitrary information (
 *                       app_start_timer() call to the timeout handler.
 */
static void battery_level_meas_timeout_handler(void * p_context)
{
    UNUSED_PARAMETER(p_context);
    NRF_LOG_INFO("Battery Level timeout event");

    // Only send the battery level update if we are connected
    if (m_conn_handle != BLE_CONN_HANDLE_INVALID)
    {
        battery_level_update();
    }
}
```

```c
/**@brief Function for updating the Battery Level measurement*/
static void battery_level_update(void)
{
    ret_code_t err_code;

    uint8_t  battery_level;
    uint16_t vbatt;              // Variable to hold voltage reading
    battery_voltage_get(&vbatt); // Get new battery voltage

    battery_level = battery_level_in_percent(vbatt);          //Transform the
    printf("ADC result in percent: %d\r\n", battery_level);

    err_code = ble_bas_battery_level_update(&m_bas, battery_level, m_conn_hand
    if ((err_code != NRF_SUCCESS) &&
        (err_code != NRF_ERROR_INVALID_STATE) &&
        (err_code != NRF_ERROR_RESOURCES) &&
        (err_code != BLE_ERROR_GATTS_SYS_ATTR_MISSING)
       )
    {
        APP_ERROR_HANDLER(err_code);
    }
}
```

- Next, we need to add the timer that triggers the battery level read operation.

  We add this to the **timers_init()** function:

```c
/**@brief Function for the Timer initialization.
 *
 * @details Initializes the timer module. This creates and starts application
 */
static void timers_init(void)
{
    // Initialize timer module.
    ret_code_t err_code = app_timer_init();
    APP_ERROR_CHECK(err_code);

    // Create timers.
    err_code = app_timer_create(&m_battery_timer_id,
                                APP_TIMER_MODE_REPEATED,
                                battery_level_meas_timeout_handler);
    APP_ERROR_CHECK(err_code);
}
```

- Add a function to start the timer, and then call it from **main()**

```c
```

```c
/**@brief Function for starting application timers.
 */
static void application_timers_start(void)
{
    uint32_t err_code;

    // Start application timers.
    err_code = app_timer_start(m_battery_timer_id, BATTERY_LEVEL_MEAS_INTERVAL
    APP_ERROR_CHECK(err_code);
}
```

- Enable the SAADC (Analog to digital converter), needed for reading the battery voltage level. Open **sdk_config.h** and enable the SAADC in **two locations**:

```c
// <e> NRFX_SAADC_ENABLED - nrfx_saadc - SAADC peripheral driver
//==========================================================
#ifndef NRFX_SAADC_ENABLED
#define NRFX_SAADC_ENABLED 1
#endif
```

```c
// <e> SAADC_ENABLED - nrf_drv_saadc - SAADC peripheral driver - legacy layer
//==========================================================
#ifndef SAADC_ENABLED
#define SAADC_ENABLED 1
#endif
```

- Add the SAADC driver file. Right click on “**nRF_Drivers**” and click on “**Add Existing File**“. Then navigate to “**nRF5_SDK_current/modules/nrfx/drivers/src/**” and select the file “**nrfx_saadc.c**“

Initializing the Services

Finally, we need to initialize the LED and Battery Services in the main file (**main.c**).

To do that, we implement the following function:

```c
```

```
/**@brief Function for initializing services that will be used by the application.
 */
static void services_init(void)
{
    uint32_t       err_code;
    ble_bas_init_t bas_init;
    ble_led_service_init_t led_init;

    nrf_ble_qwr_init_t qwr_init = {0};

    // Initialize Queued Write Module.
    qwr_init.error_handler = nrf_qwr_error_handler;

    err_code = nrf_ble_qwr_init(&m_qwr, &qwr_init);
    APP_ERROR_CHECK(err_code);

    // 1. Initialize the LED service
    led_init.led_write_handler = led_write_handler;

    err_code = ble_led_service_init(&m_led_service, &led_init);
    APP_ERROR_CHECK(err_code);

    // 2. Initialize Battery Service.
    memset(&bas_init, 0, sizeof(bas_init));

    // Here the sec level for the Battery Service can be changed/increased.
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&bas_init.battery_level_char_attr_md.cccd_write_
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&bas_init.battery_level_char_attr_md.read_perm);
    BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&bas_init.battery_level_char_attr_md.write_

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&bas_init.battery_level_report_read_perm);

    bas_init.evt_handler          = NULL;
    bas_init.support_notification = true;
    bas_init.p_report_ref         = NULL;
    bas_init.initial_batt_level   = 100;

    err_code = ble_bas_init(&m_bas, &bas_init);
    APP_ERROR_CHECK(err_code);
}
```

The function initializes both Services as well as defines the permissions of the Battery Service and Battery Level Characteristic.

Now, this needs to gets called from the main function **main():**

```
    gap_params_init();
    gatt_init();
    services_init();
    advertising_init();
    conn_params_init();
```

Flashing and debugging

Before we build and flash our program to the development kit (nRF52840), we want to enable logging first so we can better follow the state of events and the state of our application.

To get this working:

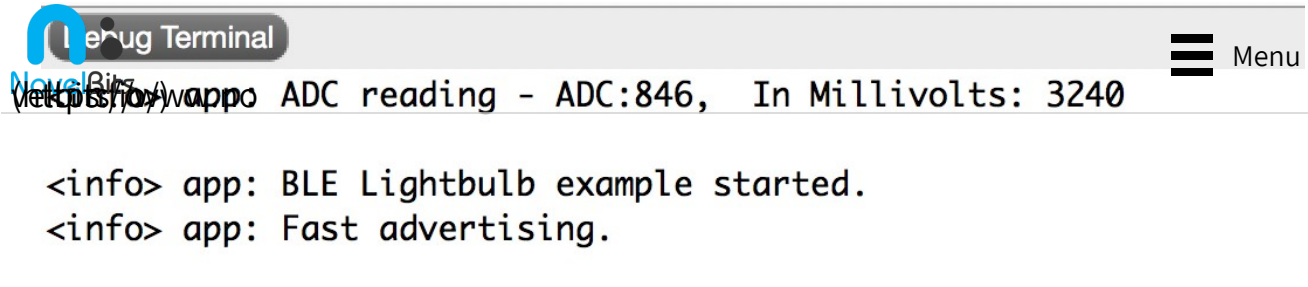
- Open **sdk_config.h** and make sure the following macros are set to the correct debug level (4). This level will enable all debug statements in the debug terminal within SES when you run your application.

```c
// <o> NRF_LOG_DEFAULT_LEVEL  - Default Severity level

// <0=> Off
// <1=> Error
// <2=> Warning
// <3=> Info
// <4=> Debug

#ifndef NRF_LOG_DEFAULT_LEVEL
#define NRF_LOG_DEFAULT_LEVEL 4
#endif
```

Now, we can flash the application to the development kit.

To do this:

- Right click on “**Project ‘ble_lightbulb’**“
- Select “**Build**“
- Right-click again on “**Project ‘ble_lightbulb’**“
- Select “**Debug → Start Debugging**“
- The application will start and stop at **main()** waiting for you to continue execution
- You should now see the output in the debug terminal similar to the following:



(https://www.novelbits.io/wp-content/uploads/2018/05/Screen-Shot-2018-05-29-at-1.19.39-AM.png)

Testing

Our next and final step is to test our application and make sure it's working fine. For this, we will use a mobile phone app that acts as a BLE Central.

For this example, we'll be using an app called **LightBlue** (available for iOS (https://itunes.apple.com/us/app/lightblue-explorer/id557428110?mt=8) and Android (https://play.google.com/store/apps/details?id=com.punchthrough.lightblueexplorer&hl=en_US)).

Here's a video showing the application running on the development kit:

Summary

In this post, we went over how to build a full BLE peripheral application that you can use as a starting point for developing any BLE application on the nRF52 platform. We went over:

- Setting up the Project and Solution file for the application
- Fully implementing a BLE application that allows you to turn ON/OFF an LED on the nRF52840 development kit
- Building, flashing, and debugging the application on the nRF52840 development kit

To be notified when future blog posts are published here on the Novel Bits blog, *be sure to enter your email address in the form below!*

If you would like to download the code used in this post, please enter your email address in the form below. You'll get a .zip containing all the source code, and I will also send you a **FREE 9-page Report on the Essential Bluetooth Developer Tools**. In addition to that, you will receive **exclusive content, tips, and tricks** that I don't post to the blog!

Email address:

Your email address

Download the code!

Posted in BLE technology (https://www.novelbits.io/category/ble/), Tutorials (https://www.novelbits.io/category/tutorials/) and tagged BLE (https://www.novelbits.io/tag/ble/), nRF52 (https://www.novelbits.io/tag/nrf52/), Smart lighbulb (https://www.novelbits.io/tag/smart-lighbulb/), Tutorial (https://www.novelbits.io/tag/tutorial/)

← The complete cross-platform nRF development tutorial (https://www.novelbits.io/cross-platform-nrf-development-tutorial/)

How to build the simplest nRF52 BLE Central (Lightbulb use case) → (https://www.novelbits.io/ble-central-lightbulb-remote-control/)

Jim on June 9, 2018 at 7:23 pm

Nice work, but I think you need to double check your code samples: I found several missing definitions and the source for the led_service.c file does not match the sections you split out for explanations (e.g. check for LED2 vs LED1), making me wonder if you used the wrong versions at some point.

Likewise, you missed documenting a few steps in editing the project file,like making sure the current directory is in the include path.

That said, I really appreciate an extensive example. Thanks.

Reply

Mohammad Afaneh (https://www.novelbits.io) on June 10, 2018 at 1:37 am

Jim, thanks for the feedback.

I updated the listed source code to match the downloadable code — I had at some point fixed some issues (such as the LED1 vs. LED2 mismatch) in the source code but forgot to update the post to include the changes.

I don't see any missing definitions locally on my computer, can you tell me which ones you found missing so I could include them? Also, I didn't have to do anything with the include path, can you list the changes you had to make to the project file?

Thanks again!

Reply

Jawed Sayed on June 24, 2018 at 3:01 pm

hi Mohammad, I was excited to find your article and step-by-step tutorial as I am new to Nordic BLE.
Unfortunately I am stuck in your "How to build the simplest nRF52 BLE Peripheral application (Lightbulb"
use case) lesson. I downloaded your code. Created the similar environment per your lesson. But cannot
compile your exact code. I am getting following errors and need some help….Thank you
I did not do any modification to your code or environment setting but duplicated exactly per your lesson

————————————————————————————————————

Building 'ble_lightbulb' from solution 'ble_lightbulb_pca10056_s140' in configuration 'Debug'

1> Compiling ‘nrf_log_backend_serial.c’

2> Compiling ‘nrf_log_backend_rtt.c’

1> In file included from C:\Users\jsaye\Desktop\nordic-BLE-5-Dev\nRF5_SDK_current\components\libraries\experimental_log\src\nrf_log_backend_serial.c:40:0:

1> ../nRF5_SDK_current/components/libraries/util/sdk_common.h:56:10: fatal error: sdk_config.h: No such file or directory

1> compilation terminated.

3> Compiling ‘nrf_log_backend_uart.c’

2> In file included from C:\Users\jsaye\Desktop\nordic-BLE-5-Dev\nRF5_SDK_current\components\libraries\experimental_log\src\nrf_log_backend_rtt.c:40:0:

2> ../nRF5_SDK_current/components/libraries/util/sdk_common.h:56:10: fatal error: sdk_config.h: No such file or directory

2> compilation terminated.

3> In file included from C:\Users\jsaye\Desktop\nordic-BLE-5-Dev\nRF5_SDK_current\components\libraries\experimental_log\src\nrf_log_backend_uart.c:40:0:

3> ../nRF5_SDK_current/components/libraries/util/sdk_common.h:56:10: fatal error: sdk_config.h: No such file or directory

3> compilation terminated.

Build failed

Reply

Mohammad Afaneh (https://www.novelbits.io) on June 25, 2018 at 10:51 pm

Hi Jawed,

Did you check if the sdk_config.h file exists in the main lightbulb folder?

What SDK version are you using?

Reply

Jawed Sayed on June 25, 2018 at 1:20 pm

Hi Mohammad, Looks like my previous comment was removed, Any reason or so? -thx

Reply

Jawed Sayed on June 26, 2018 at 1:21 am

Hi Mohammad, Thank you for your quick response.

1. I am using nR5_SDK_15.0.0_a53641a

2. Yes sdk_config.h file exists under the main lightbulb folder.

Here is what I did:

a. create a directory named nordic_BLE_Dev

b. downloaded the SDK_15.0,0_a53641a under nordic_BLE_Dev directory

c. renamed the SDK to nRF5_SDK_current

d. downloaded lightblub from your code link, unzipped it and copied it under nordic_BLE_Dev

e. As it is exact downloaded copy of your code – it has sdk_config.h under the main lightbulb directory

.f. run the project file and got errors. I tried it again and still got same errors.

Building 'ble_lightbulb' from solution 'ble_lightbulb_pca10056_s140' in configuration 'Debug'

1> Assembling 'thumb_crt0.s'

2> Compiling 'nrf_log_backend_rtt.c'

3> Compiling 'nrf_log_backend_serial.c'

2> In file included from C:\Users\jsaye\Desktop\nordic_BLE_Dev\nRF5_SDK_current\components\libraries\experimental_log\src\nrf_log_backend_rtt.c:40:0:

2> ../nRF5_SDK_current/components/libraries/util/sdk_common.h:56:10: fatal error: sdk_config.h: No such file or directory

2> compilation terminated.

4> Compiling 'nrf_log_backend_uart.c'

3> In file included from C:\Users\jsaye\Desktop\nordic_BLE_Dev\nRF5_SDK_current\components\librar

es\experimental_log\src\nrf_log_backend_serial.c:40:0:

../nRF5_SDK_current/components/libraries/util/sdk_common.h:56:10: fatal error: sdk_config.h: No such file or directory

3> compilation terminated.

4> In file included from C:\Users\jsaye\Desktop\nordic_BLE_Dev\nRF5_SDK_current\components\libraries\experimental_log\src\nrf_log_backend_uart.c:40:0:

4> ../nRF5_SDK_current/components/libraries/util/sdk_common.h:56:10: fatal error: sdk_config.h: No such file or directory

4> compilation terminated.

Build failed

Reply

Mohammad Afaneh (https://www.novelbits.io) on June 26, 2018 at 9:21 am

Hi Jawed,

The only thing I can think of is that the include path for the project does not include the root path for the project. Can you open the solution file (*.emProject file) in a text editor and check if the following exists:

c_user_include_directories="./

You can also try changing it to use the full absolute path instead of the

relative one. Also, make sure the file is included in the Project in SES.

(https://www.no

-Mohammad

Reply

Carl Looper on August 17, 2018 at 4:11 pm

I added the following to the list of directories and the compiler was able to find the sdk_config.h file and sucessfully compile.

c_user_include_directories = “../ble_lightbulb;

Reply

Mohammad Afaneh on August 17, 2018 at 4:19 pm

Thanks, Walter and Carl.

The issue seems to occur on Windows only. Here’s the fix: (I’ll fix this in the downloadable project as well)

Modify the Project file (ble_lightbulb_pca10056_s140.emProject) to

“ c_user_include_directories=”.;../nRF5_SDK_current/component

”

Instead of

“ c_user_include_directories=”./;../nRF5_SDK_current/componen
t”

So, basically just remove the trailing “/” after the “.”. For some reason, Windows does not like it.

Thanks again.

-Mohammad

Reply

Jawed Sayed on June 26, 2018 at 6:40 pm

Hi Mohammad, Thanks again.

What you mentioned in your email is not possible because my project is exact copy of yours. So you would have see these issues before I. Anyway.. I was able to move forward and compile your project without errors.

To solve my previous problem, I had to:

a. create a sub-directory call “config” same level as my main.c file.

b. move the sdk_config.h file in the new config directory.

c. add “./config” in the emProject file.

I have tried but I don’t know why sdk_config.h needs to be under the config

directory.

It does not work without it, even after you change all the paths (2 path). Even if you hard code the paths.

I am okay with a config directory infact I prefer it. so i call this issue resolved.

Now I have issues in the main.c file, linker cannot find functions. (see below) I will work on it later today

and let you know, most probably some additional path issues in my environment.

Do you know – when I installed the Sagger Embedded studio, do I need to set some paths in window 10?

Thanks

…jsayed

— Errors ——

1> Output/ble_itaalaLock Debug/Obj/main.o:C:\Users\jsaye\Desktop\nordic_BLE_Dev\ble_itaalaLock/main.c:140: undefined reference to `ble_bas_on_ble_evt'`

`1> Output/ble_itaalaLock Debug/Obj/main.o: In function` services_init':

1> C:\Users\jsaye\Desktop\nordic_BLE_Dev\ble_itaalaLock/main.c:292: undefined reference to `ble_bas_init'`

`1> Output/ble_itaalaLock Debug/Obj/main.o: In function` battery_level_update':

1> C:\Users\jsaye\Desktop\nordic_BLE_Dev\ble_itaalaLock/main.c:670: undefined reference to `ble_bas_battery_level_update'`

`1> Output/ble_itaalaLock Debug/Obj/battery_voltage.o: In function`

nrf_drv_saadc_init':

C:\Users\jsaye\Desktop\nordic_BLE_Dev\ble_itaalaLock/../nRF5_SDK_current/integration/nrfx/legacy/nrf_drv_saadc.h:134: undefined reference to `nrfx_saadc_init'

1> Output/ble_itaalaLock Debug/Obj/battery_voltage.o: In function battery_voltage_init':

1> C:\Users\jsaye\Desktop\nordic_BLE_Dev\ble_itaalaLock\Battery Level/battery_voltage.c:82: undefined reference to `nrfx_saadc_channel_init'

1> C:\Users\jsaye\Desktop\nordic_BLE_Dev\ble_itaalaLock\Battery Level/battery_voltage.c:85: undefined reference to nrfx_saadc_buffer_convert'

1> C:\Users\jsaye\Desktop\nordic_BLE_Dev\ble_itaalaLock\Battery Level/battery_voltage.c:88: undefined reference to `nrfx_saadc_sample'

1> Output/ble_itaalaLock Debug/Obj/battery_voltage.o: In function battery_voltage_get':

1> C:\Users\jsaye\Desktop\nordic_BLE_Dev\ble_itaalaLock\Battery Level/battery_voltage.c:97: undefined reference to `nrfx_saadc_is_busy'

1> C:\Users\jsaye\Desktop\nordic_BLE_Dev\ble_itaalaLock\Battery Level/battery_voltage.c:99: undefined reference to nrfx_saadc_buffer_convert'

1> C:\Users\jsaye\Desktop\nordic_BLE_Dev\ble_itaalaLock\Battery Level/battery_voltage.c:102: undefined reference to `nrfx_saadc_sample'

Build failed

Mohammad Afaneh (https://www.novelbits.io) on June 29, 2018 at 4:12 pm

> Hi Jawed,
>
> Unfortunately, I do not have a Windows machine to test this with at the moment. You may find some helpful instructions here though:
> https://www.segger.com/downloads/embedded-studio/EmbeddedStudio_Manual (https://www.segger.com/downloads/embedded-studio/EmbeddedStudio_Manual)
> https://www.youtube.com/watch?v=YZouRE_Ol8g&list=PLx_tBuQ_KSqGHmzdEL2GWEOeix-S5rgTV (https://www.youtube.com/watch?v=YZouRE_Ol8g&list=PLx_tBuQ_KSqGHmzdEL2GWEOeix-S5rgTV)

Reply

Walter on August 17, 2018 at 1:35 am

> Same problem

Building ‘ble_lightbulb’ from solution ‘ble_lightbulb_pca10056_s140’ in configuration ‘Debug’

1> Compiling ‘nrf_log_backend_serial.c’

2> Compiling ‘nrf_log_backend_rtt.c’

1> In file included from C:\BLE Projects\nRF5_SDK_current\components\libraries\experimental_log\src\nrf_log_backend_serial.c:40:0:

1> ../nRF5_SDK_current/components/libraries/util/sdk_common.h:56:10: fatal error: sdk_config.h: No such file or directory

1> compilation terminated.

3> Compiling ‘nrf_log_backend_uart.c’

2> In file included from C:\BLE Projects\nRF5_SDK_current\components\libraries\experimental_log\src\nrf_log_backend_rtt.c:40:0:

2> ../nRF5_SDK_current/components/libraries/util/sdk_common.h:56:10: fatal error: sdk_config.h: No such file or directory

2> compilation terminated.

3> In file included from C:\BLE Projects\nRF5_SDK_current\components\libraries\experimental_log\src\nrf_log_backend_uart.c:40:0:

3> ../nRF5_SDK_current/components/libraries/util/sdk_common.h:56:10: fatal error: sdk_config.h: No such file or directory

3> compilation terminated.

Build failed

Walter on August 17, 2018 at 5:18 pm

Modify the Project file (ble_lightbulb_pca10056_s140.emProject) to

“ c_user_include_directories=”.;../nRF5_SDK_current/component”

Instead of

“ c_user_include_directories=”./;../nRF5_SDK_current/component”

So, basically just remove the trailer “/” after the “.”. For some reason, Windows does not like it.

Reply

rahmat dwi putra on October 15, 2018 at 1:12 am

Hi,

Thanks for the code. I tried a few things extra in this exercise like adding LED3 and LED4 into the code. I duplicated the format to initiate the LED2 for LED3 and LED4. The code compiled however The service did not come up on the app. Is there something I did wrong? I literally just copied the format for LED 3 and LED 4.

Thanks,

Reply

Bo Chen on November 19, 2018 at 9:51 pm

Hi,

Since Nordic published the new version of SDK (15.2.0), I find a lot of problem while trying to compile the code downloaded from your link. Mainly two problems : 1) directory path error, which is may caused by the path changing of the SDK; 2) Function and structure elements name error, which is also may caused by the declaration/definition differences between the new version SDK and the old one you used. I cannot attach the bug logout here because the number of errors is too too many. Maybe the project can work before, but I really spend a lot of time on fixing the bugs, and finally give up because I can’t quiet understand what the code try to do whiling using the SDK especially with so many directory path and definition errors . I’m a new guy for using the nRF52840 kit, and really thanks for your instruction of how to start a simple BLE application. If convenient, please update the source code with new version SDK (I cannot find the old version 15.0.0 from the website), or combine the code and the SDK used for the code in one file. Thanks a lot.

Best

Reply

Mohammad Afaneh on November 19, 2018 at 9:58 pm

Hi Bo,

Thanks for your feedback. I'll work on updating the examples for SDK version 15.2.0. In the meantime, you can actually download the 15.0.0 SDK at https://developer.nordicsemi.com/nRF5_SDK/nRF5_SDK_v15.x.x/ (https://developer.nordicsemi.com/nRF5_SDK/nRF5_SDK_v15.x.x/)

Thanks,

Mohammad

Reply

Robert on November 22, 2018 at 3:26 pm

Hi Mohammad,

I have project code ported to the current version of Nordic SDK (15.2.0). I can share it on my Github account.

Mohammad Afaneh on November 22, 2018 at 3:32 pm

Thanks, Robert! That would be great!

Reply

Robert on November 22, 2018 at 3:49 pm

The code is available in the repo:

https://github.com/gruberski/ble_lightbulb

(https://github.com/gruberski/ble_lightbulb)

Mohammad Afaneh on November 25, 2018 at 4:38 pm

Thanks for sharing the code, Robert!

Leave a Comment

Comment

Name (required)

Email (will not be published) (required)

Website

Submit Comment