NORDIC techTOUR

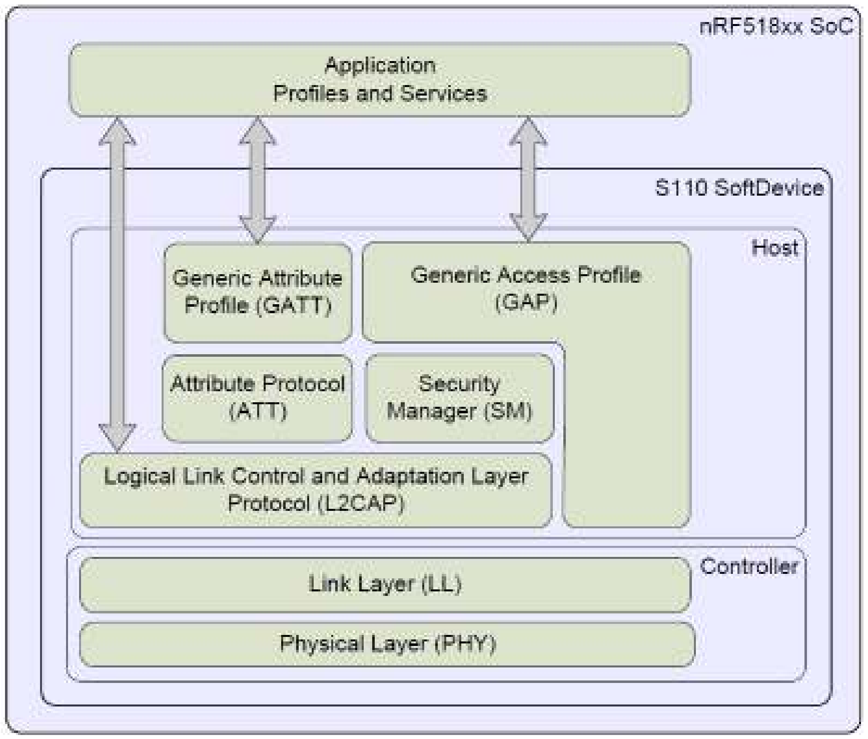
Building a GATT/GAP Profile

NORDIC SEMICONDUCTOR

How a BLE application works

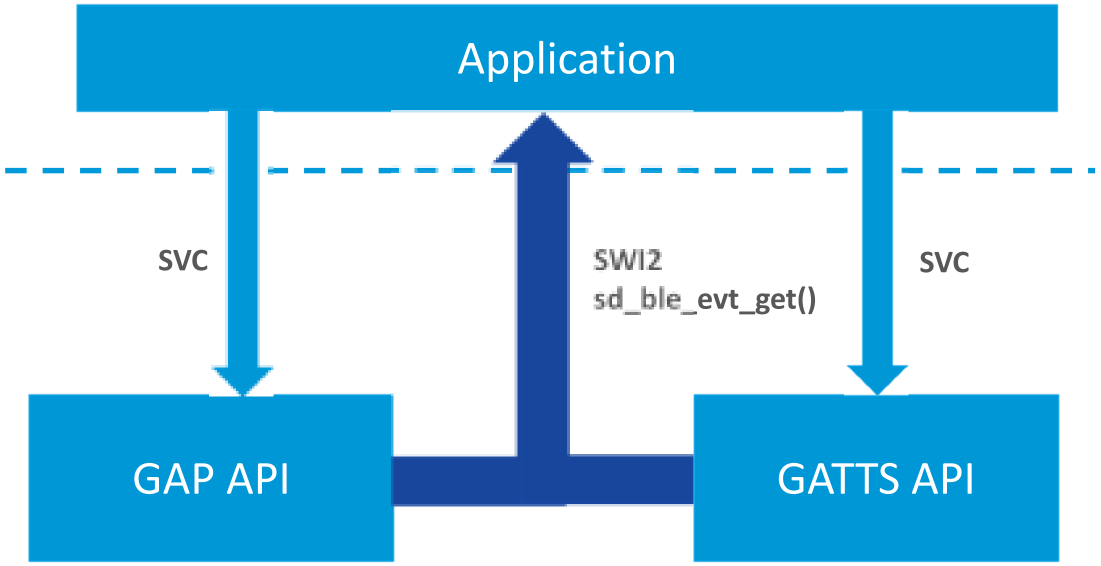
How a BLE application works ?

- SoftDevice is a protocol stack solution
  - that runs in a protected code area
  - Accompanying protected RAM area.
- SoftDevice is a precompiled and prelinked HEX file
  - Independent from the application
  - Can be programmed separately.


nRF518xx SoC
Application Profiles and Services
S110 SoftDevice
Host
Generic Attribute Profile (GATT)
Generic Access Profile (GAP)
Attribute Protocol (ATT)
Security Manager (SM)
Logical Link Control and Adaptation Layer Protocol (L2CAP)
Controller
Link Layer (LL)
Physical Layer (PHY)

Bluetooth® low energy API

- Generic Access Profile (GAP)
- Generic Attribute Profile Server
  - (GATTS)
- API calls as **S**uper**V**isor **C**alls
  - Switches Core to SV priority
  - Each SV Call numbered
- Events as **S**oft**W**are **I**nterrupts (
  - Always through SWI2 (軟体中断)
  - Interrupt priority: Application Low
  - `sd_ble_evt_get()- callback` (回调函数)
    - For all BLE events
    - From ISR or main context



BLE API: GAP SVC

- GAP Service (built in to the stack)
  - **`sd_ble_gap_device_name_set(security, name)`**
    - Sets the device name and security mode for the device name characteristic
  - **`sd_ble_gap_appearance_set(appearance)`**
    - Describes what our device does to central peers
  - **`sd_ble_gap_ppcp_set(ppcp)`**
    - Defines the connection parameters
  - …
- GAP Advertisement
  - **`sd_ble_gap_adv_data_set(adv_data, ad_len, sr_data, sr_len)`**
    - Sets the advertisement data that central peers will receive
  - **`sd_ble_gap_adv_start(adv_params)`**
    - Starts sending advertisement packets over the air
  - …

BLE API: GAP Events (by using callback 回调函数)

- `BLE_GAP_EVT_CONNECTED {conn_handle, peer_addr, conn_params}`
  - A central peer has established a physical connection
- `BLE_GAP_EVT_DISCONNECTED {conn_handle, reason}`
  - The connection has been terminated, locally or remotely
- `BLE_GAP_EVT_CONN_PARAM_UPDATE {conn_params}`
  - A connection parameter update procedure has completed
- `BLE_GAP_EVT_TIMEOUT {source}`
  - A procedure has timed out (advertisement, security, …)
- . . .

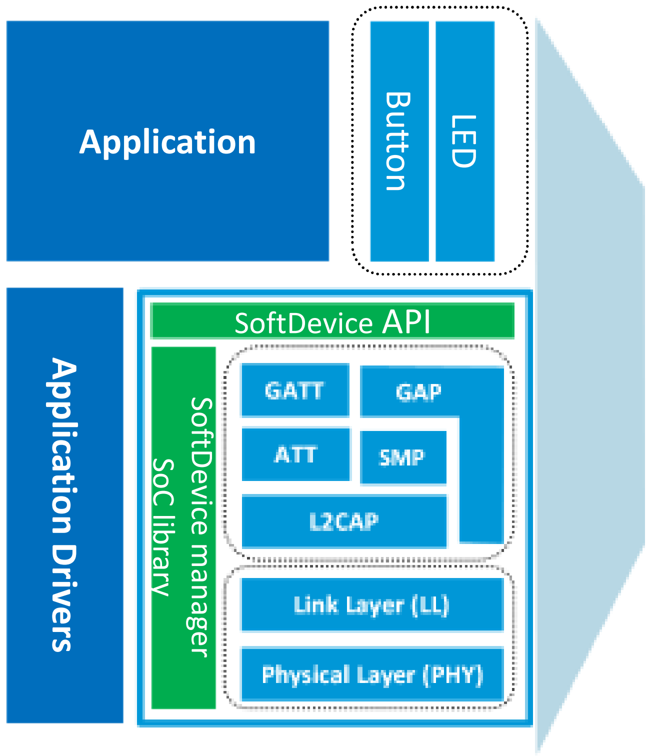
BLE API: GATTS SVC

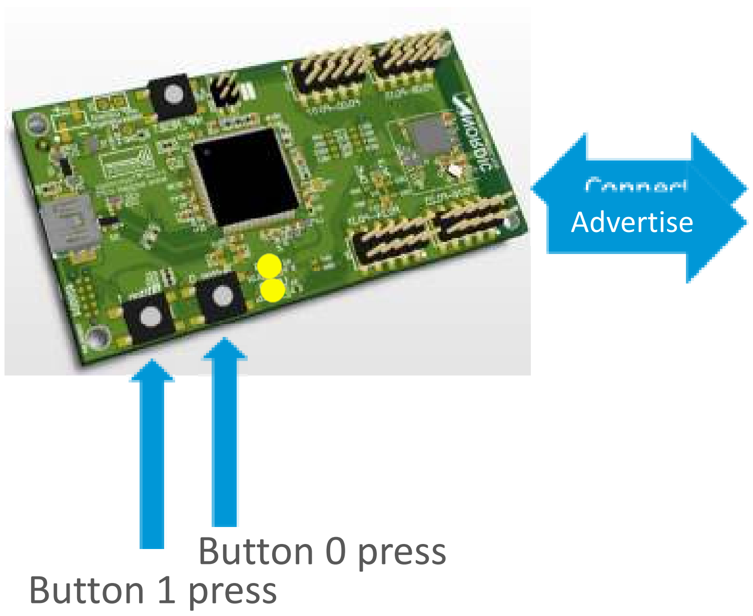
- ATT Table Population
  - **`sd_ble_gatts_service_add(type, UUID, out_handle)`**
    - Adds an empty Service to the ATT Table
  - **`sd_ble_gatts_characteristic_add(svc_handle, md, value, out_handles)`**
    - Adds a Characteristic to the referenced service
- ATT Table Local Access
  - **`sd_ble_gatts_value_set(handle, offset, len, value)`**
    - Sets the value of any particular attribute
  - **`sd_ble_gatts_value_get(handle, offset, len, value)`**
    - Gets the value of any particular attribute
- Server Initiate
  - **`sd_ble_gatts_hvx(conn_handle, params)`**
    - Sends an ATT Notification or Indication

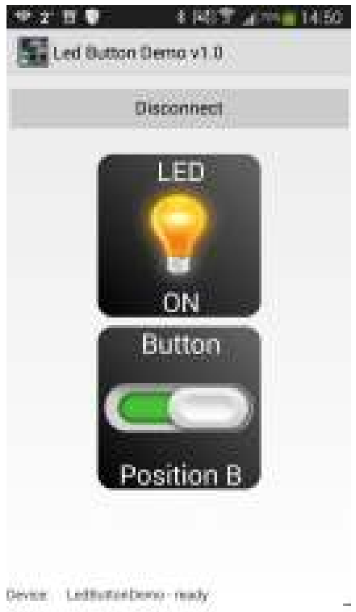
BLE API: GATTS Events

- `BLE_GATTS_EVT_WRITE {conn_handle, handle, data}`
  - An incoming client ATT Write operation has received and executed
- `BLE_GATTS_EVT_HVC {conn_handle, handle}`
  - A Handle Value Confirmation has been received from the peer
- . . .

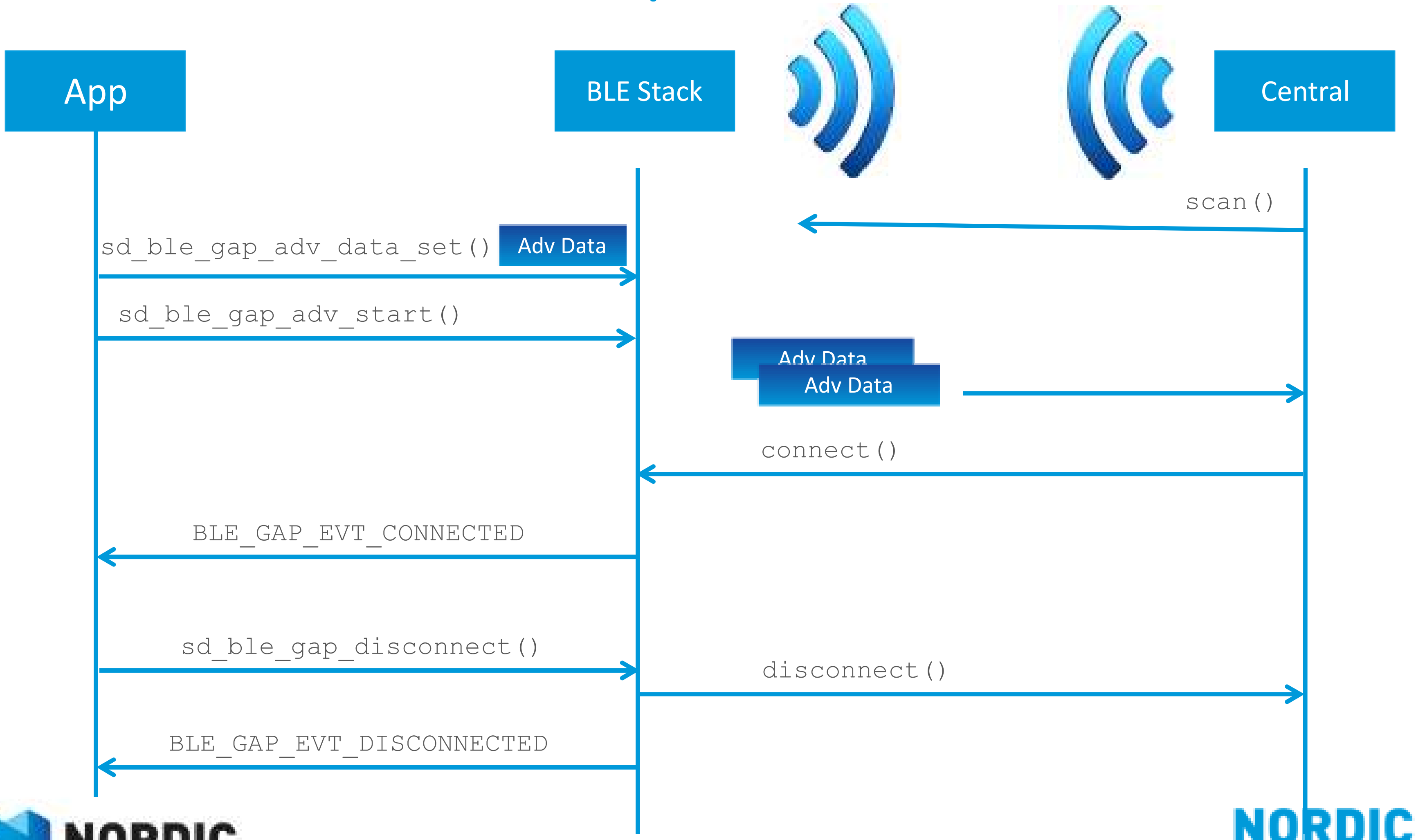
LED Button application



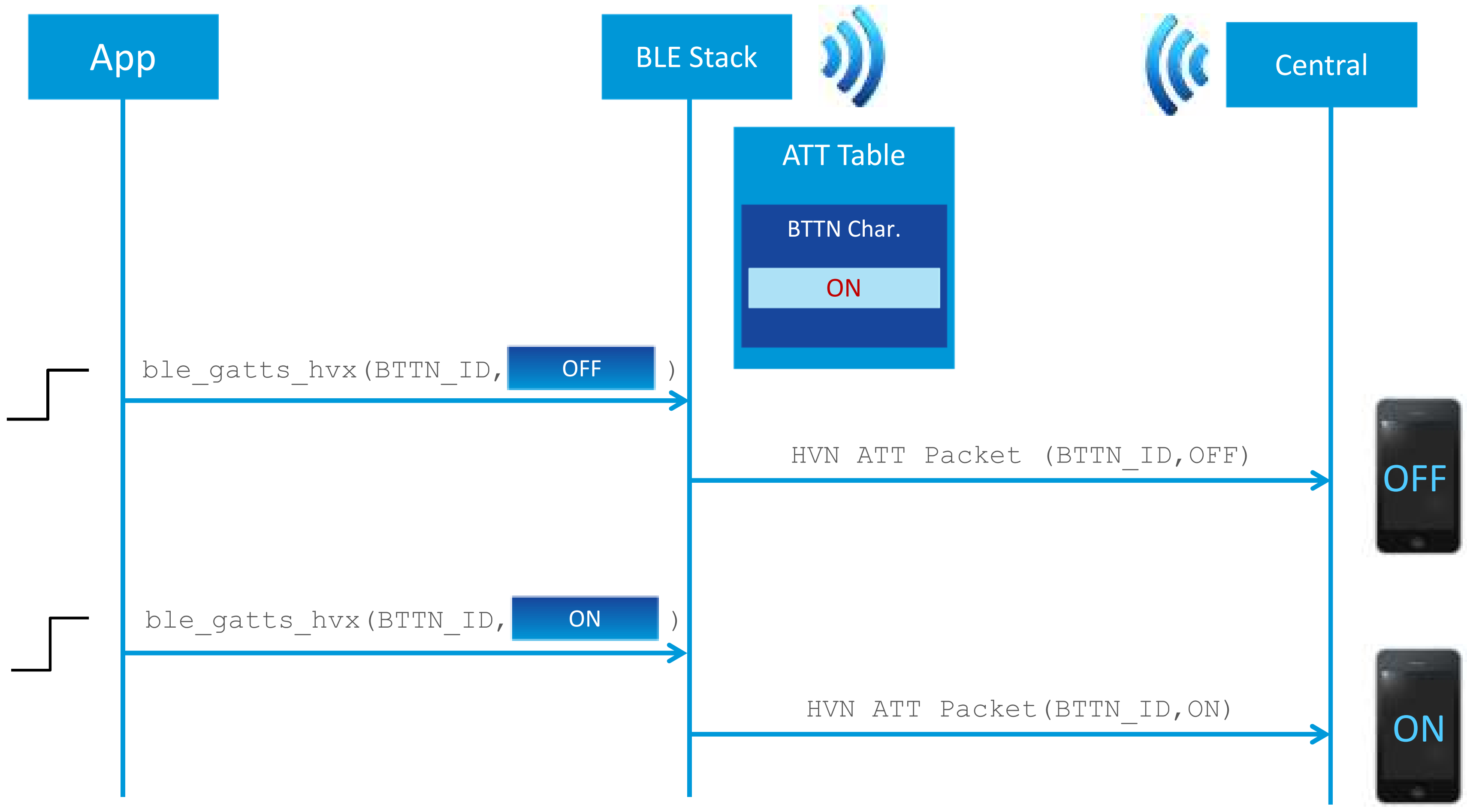




BLE API: Connection Sequence



BLE API: Handle Value Notification

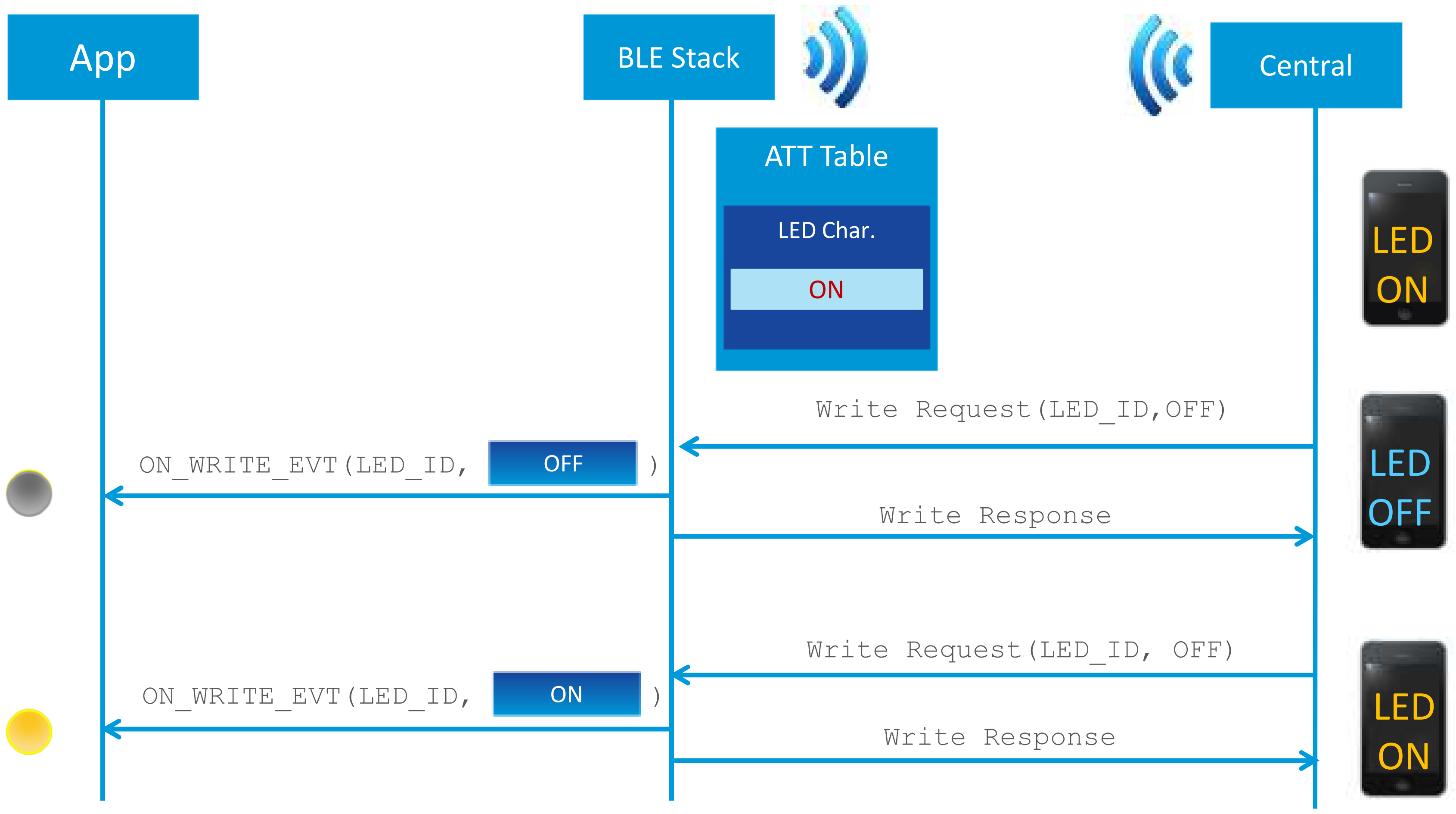


App
BLE Stack
Central
ATT Table
BTTN Char.
ON
ble_gatts_hvx(BTTN_ID, OFF)
HVN ATT Packet (BTTN_ID,OFF)
OFF
ble_gatts_hvx(BTTN_ID, ON)
HVN ATT Packet(BTTN_ID,ON)
ON

NORDIC SEMICONDUCTOR

NORDIC tech TOUR

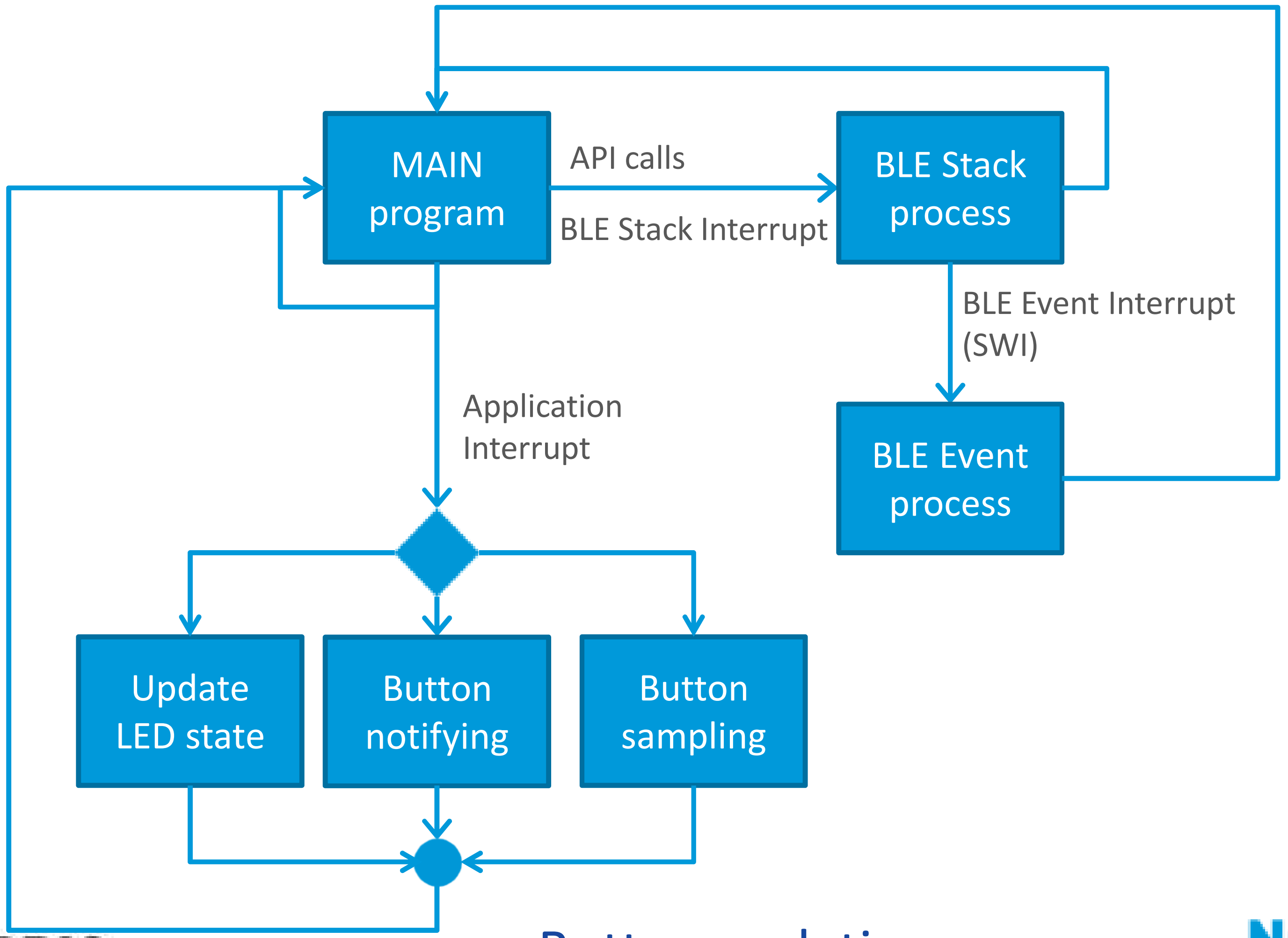
BLE API: Handle Write Commands



NORDIC SEMICONDUCTOR

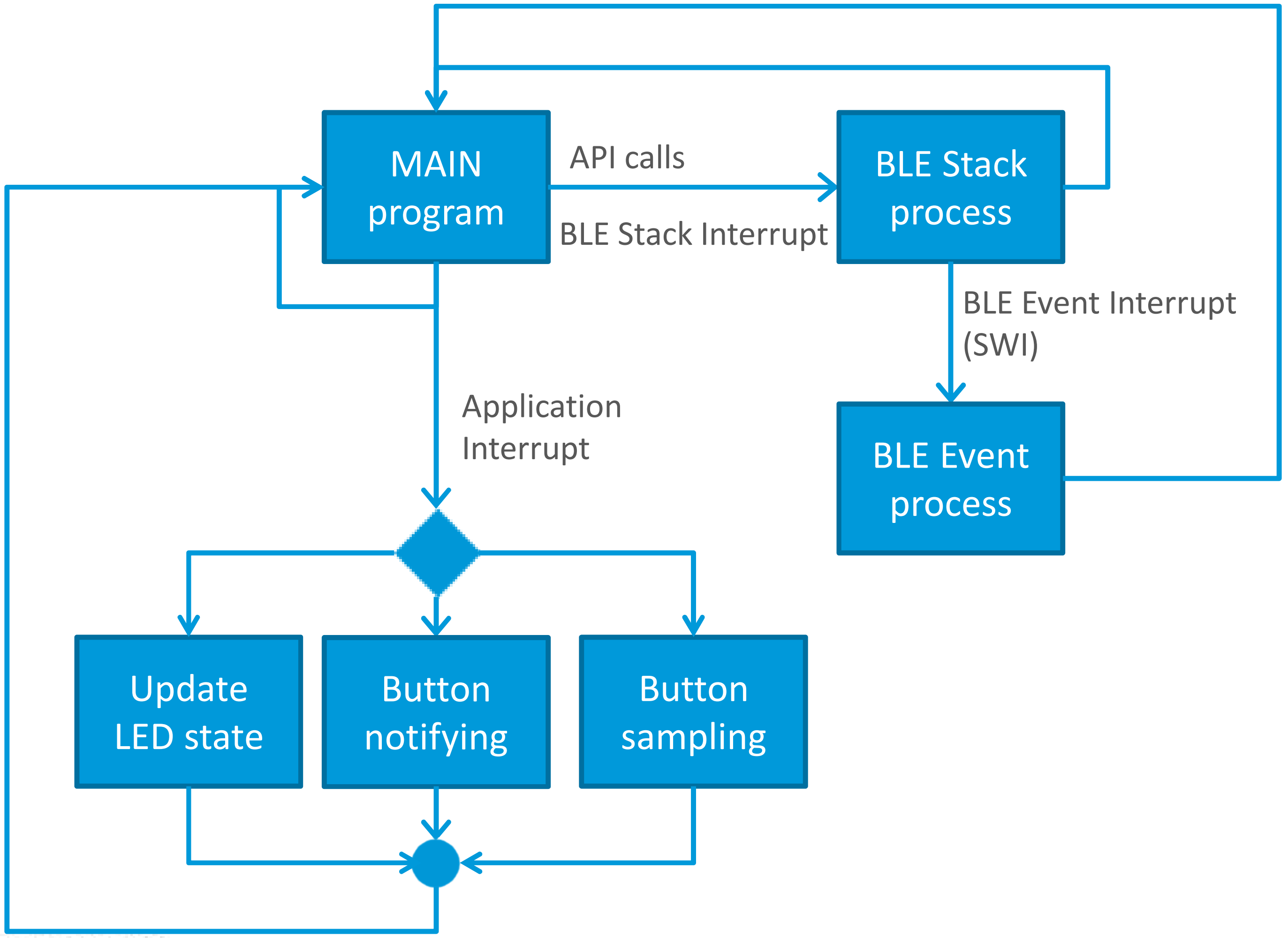
NORDIC tech TOUR

Application block diagram



Button updating

Application block diagram



LED updating

Building the data structure

- Button characteristic
  - One byte to notify the button state.
    - 0: ON
    - 1: OFF
  - Properties:
    - Read
    - Notification
  - Permission:
    - Read

- LED characteristic
  - One byte to update the LED state
    - 0: ON
    - 1: OFF
  - Properties:
    - Read
    - Write
  - Permission:
    - Read
    - Write

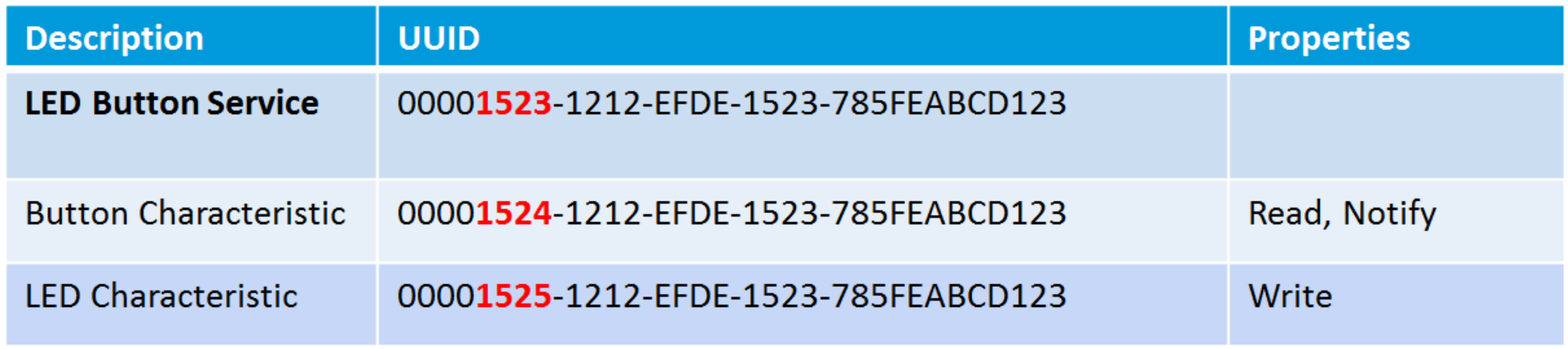
Implementing data structures

- LED Button Service structure

```
typedef struct ble_lbs_s
{
    uint16_t                        service_handle;   //handle is used for Unique ID 唯一號碼
    ble_gatts_char_handles_t        led_char_handles;
    ble_gatts_char_handles_t        button_char_handles;
    uint8_t                         uuid_type;          // 128-bit UUID or 16-bit UUID service
    uint16_t                        conn_handle;        // connection handle
    ble_lbs_led_write_handler_t led_write_handler;      // Event handler to be called when LED
                                                        characteristic is written.

} ble_lbs_t;
```

Standard versus custom services and char

- A se
- B



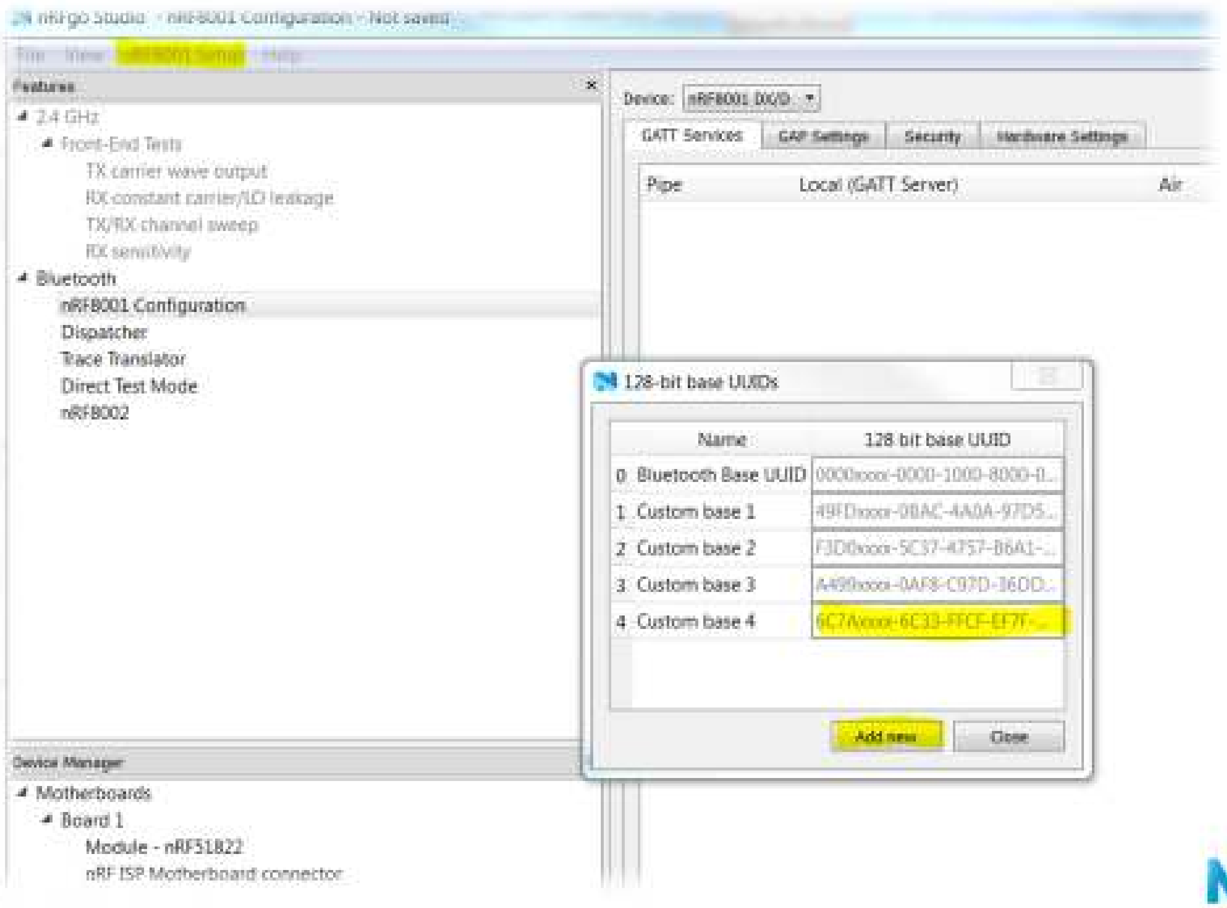
Description	UUID	Properties
LED Button Service	00001523-1212-EFDE-1523-785FEABCD123	
Button Characteristic	00001524-1212-EFDE-1523-785FEABCD123	Read, Notify
LED Characteristic	00001525-1212-EFDE-1523-785FEABCD123	Write

  -
    - 0x00001234-0000-1000-8000-00805F9B34FB
  - All Bluetooth SIG attribute will have UUID:
    - 0x0000**xxxx**-0000-1000-8000-00805F9B34FB
  - For LED Button example, an base UUID is generated:
    - 0x0000**xxxx**-1212-EFDE-1523-785FEABCD123

- Alias is the 16 bit that are not defined by the Base UUID.
  - For example Battery Service UUID is 0x180F , Battery Level char is 0x2A19, etc
  - For LED Button example
    - Service: 0x1523
    - LED characteristic: 0x1524
    - Button characteristic 0x1525

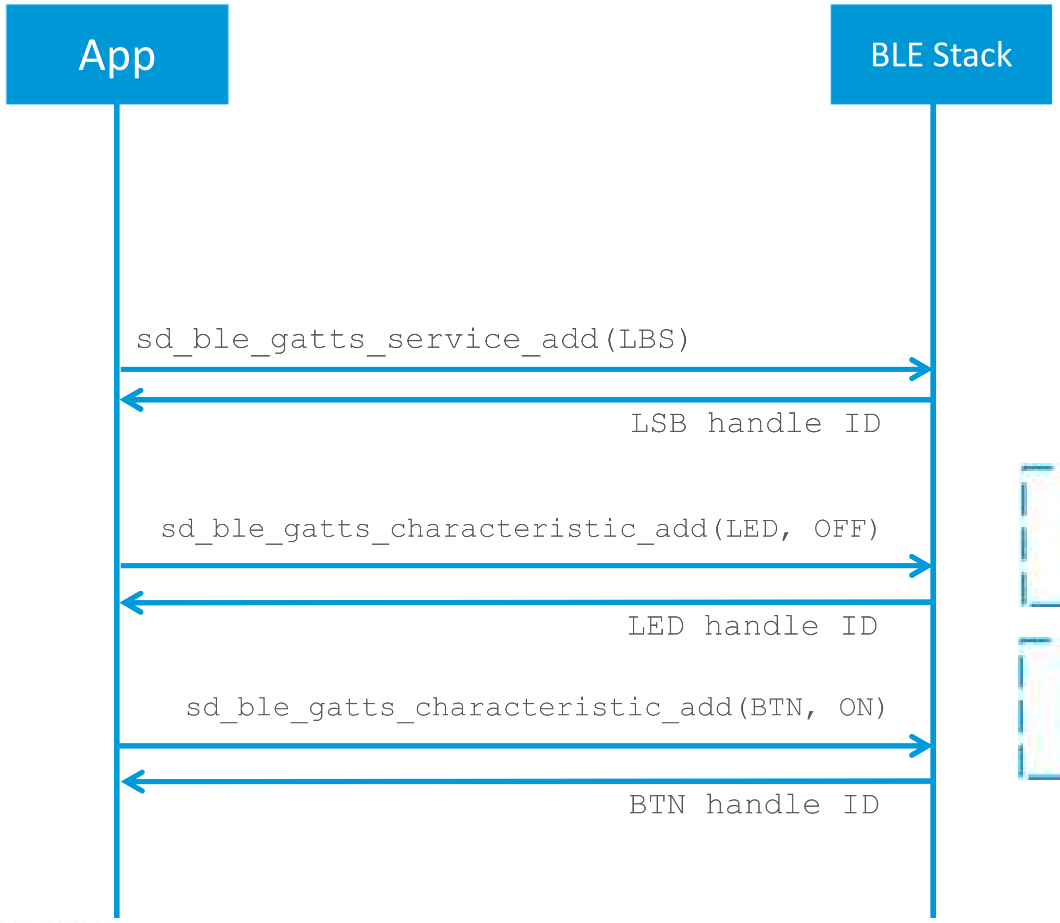
Characteristics

- How to generate 128 bit UUID using 8001 configuration tool:



	Name	128 bit base UUID
0	Bluetooth Base UUID	0000xxxx-0000-1000-8000-0...
1	Custom base 1	49FDxxxx-0BAC-4A0A-97D5...
2	Custom base 2	F3D0xxxx-5C37-4757-B6A1-...
3	Custom base 3	A499xxxx-0AF8-C97D-36DD...
4	Custom base 4	6C7Axxxx-6C33-FFCF-EF7F-...

BLE API: Service population


App
BLE Stack
sd_ble_gatts_service_add(LBS)
LSB handle ID
sd_ble_gatts_characteristic_add(LED, OFF)
LED handle ID
sd_ble_gatts_characteristic_add(BTN, ON)
BTN handle ID

Unique ID 唯一號碼

Handle	UUID	Value
0x0001	*SERVICE*	*LBS*
0x0002	*CHAR*	*LED*
0x0003	*LED*	OFF
0x0004	*CHAR*	*BTN*
0x0005	*BTN*	*ON*

GATT Initialization

- LBS Service Initialization
  - *uint32_t ble_lbs_init(ble_lbs_t * p_lbs, const ble_lbs_init_t * p_lbs_init) {*

```
    ble_uuid128_t base_uuid = LBS_UUID_BASE;
    ble_uuid.uuid = LBS_UUID_SERVICE;

    sd_ble_uuid_vs_add(&base_uuid, &p_lbs->uuid_type);

    sd_ble_gatts_service_add(BLE_GATTS_SRVC_TYPE_PRIMARY, &ble_uuid, &p_lbs->service_handle);

    button_char_add(p_lbs, p_lbs_init);

    led_char_add(p_lbs, p_lbs_init);

}
```

GATT Initialization

Properties: Read, Notification

- Button Characteristic Initialization
  - *uint32_t button_char_add(ble_lbs_t * p_lbs, const ble_lbs_init_t * p_lbs_init)*

```
{
 BLE_GAP_CONN_SEC_MODE_SET_OPEN(&cccd_md.read_perm);
 BLE_GAP_CONN_SEC_MODE_SET_OPEN(&cccd_md.write_perm);

  char_md.char_props.read   = 1;
  char_md.char_props.notify = 1;
  char_md.p_cccd_md         = &cccd_md;

  ble_uuid.type = p_lbs->uuid_type;
  ble_uuid.uuid = LBS_UUID_BUTTON_CHAR;

  BLE_GAP_CONN_SEC_MODE_SET_OPEN(&attr_md.read_perm);
  BLE_GAP_CONN_SEC_MODE_SET_NO_ACCESS(&attr_md.write_perm);
  attr_md.rd_auth    = 0;
  attr_md.wr_auth    = 0;
  attr_md.vlen       = 0;

  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_lbs->service_handle, &char_md, &attr_char_value,
                                          &p_lbs->button_char_handles);
}
```

CCCD permission 允许

Characteristic's properties and UUID

Characteristic's permissions and metadata (允许)

Assign characteristic's UUID, permissions and init value

GATT Initialization

Properties: Read, Write

- LED Characteristic Initialization
  - *uint32_t led_char_add(ble_lbs_t * p_lbs, const ble_lbs_init_t * p_lbs_init)*

```
{
    char_md.char_props.read   = 1;
    char_md.char_props.write  = 1;
    char_md.p_cccd_md         = NULL;

    ble_uuid.type = p_lbs->uuid_type;
    ble_uuid.uuid = LBS_UUID_LED_CHAR;

    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&attr_md.read_perm);
    BLE_GAP_CONN_SEC_MODE_SET_OPEN(&attr_md.write_perm);
    attr_md.rd_auth    = 0;
    attr_md.wr_auth    = 0;
    attr_md.vlen       = 0;

    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_lbs->service_handle, &char_md, &attr_char_value,
                                           &p_lbs->led_char_handles);
}
```

Characteristic's properties
No CCCD with LED char

UUID type and UUID value

Characteristic's permissions and metadata

Assign characteristic's UUID, permissions and init value

Designing the API

The LED Button service needs to: (用Notify 轉送訊號去 主機)

- Notify the central device when there button state is changed. So, we need to add a method to be called when the button state changes.

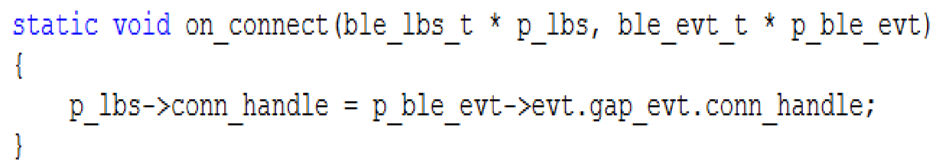
```
uint32_t ble_lbs_on_button_change(ble_lbs_t * p_lbs, uint8_t button_state)
{
        params.type = BLE_GATT_HVX_NOTIFICATION;
        params.handle = p_lbs->button_char_handles.value_handle;
        params.p_data = &button_state;
        params.p_len = &len;

        return sd_ble_gatts_hvx(p_lbs->conn_handle, &params);  // SVC GATTS API for notification
}
```

Designing the API

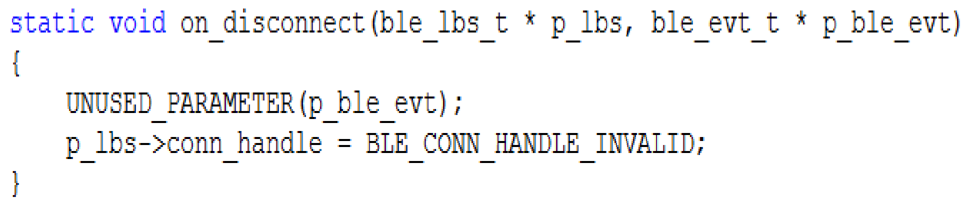
The LED Button service needs to:

- Update the LED state when a write command is received from the central device. The service need to handle the write event anc

```
void ble_lbs_on_ble_evt(ble_lbs_t * p
{
    switch (p_ble_evt->header.evt_id)
        case BLE_GAP_EVT_CONNECTED:
            on_connect(p_lbs, p_ble_e
            break;

        case BLE_GAP_EVT_DISCONN
            on_disconnect(p_lbs, p_bl

        case BLE_GATTS_EVT_WRITE:
            on_write(p_lbs, p_ble_evt);
}
```



```
static void on_connect(ble_lbs_t * p_lbs, ble_evt_t * p_ble_evt)
{
    p_lbs->conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
}
```



```
static void on_disconnect(ble_lbs_t * p_lbs, ble_evt_t * p_ble_evt)
{
    UNUSED_PARAMETER(p_ble_evt);
    p_lbs->conn_handle = BLE_CONN_HANDLE_INVALID;
}
```

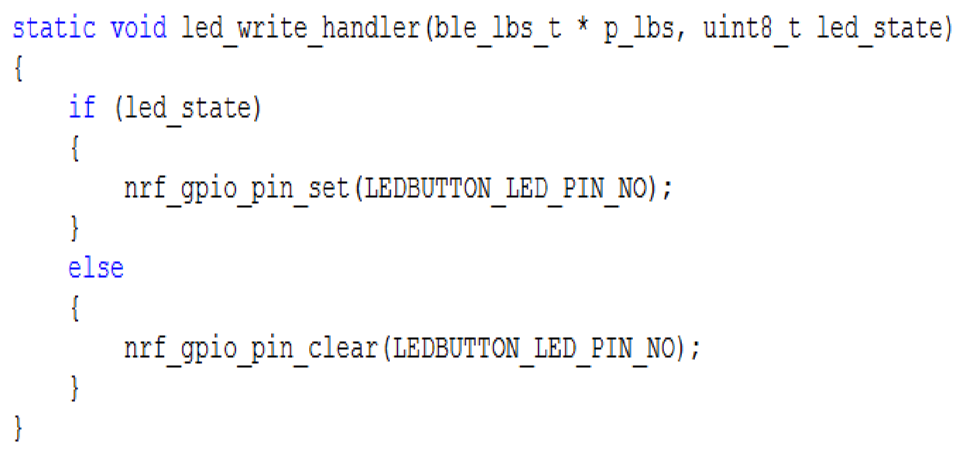
NORDIC SEMICONDUCTOR

NORDIC tech TOUR

Designing the API

The LED Button service needs to know:

- When a write command is received from the central device to update the LED state. The service need to handle the write



```
static void led_write_handler(ble_lbs_t * p_lbs, uint8_t led_state)
{
    if (led_state)
    {
        nrf_gpio_pin_set(LEDBUTTON_LED_PIN_NO);
    }
    else
    {
        nrf_gpio_pin_clear(LEDBUTTON_LED_PIN_NO);
    }
}
```

*static void on_write(ble_lbs_t * p_l*

{

*ble_gatts_evt_write_t * p_evt_*

if ((p_evt_write->handle == p_lk

(p_evt_write->len == 1) && (

{

p_lbs->led_write_handler(p_lbs, p_evt_write->data[0]);

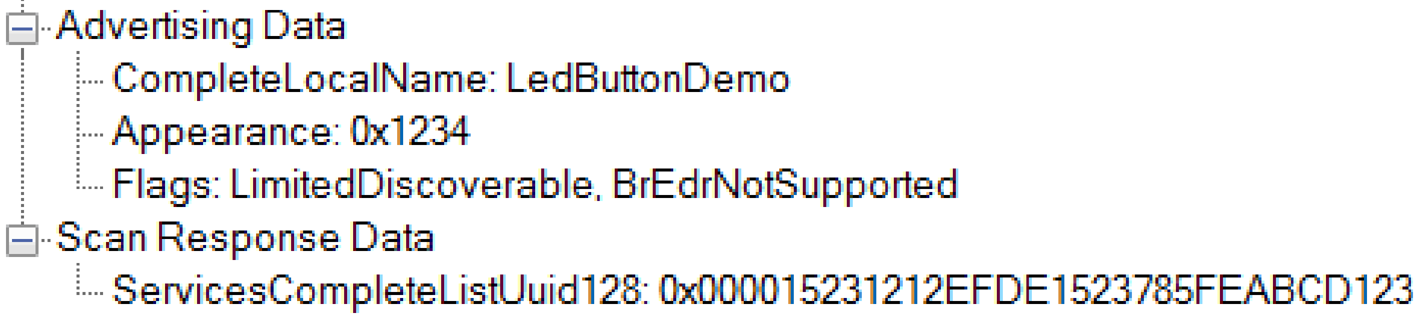
}

}

Last step: setting up Advertising packet

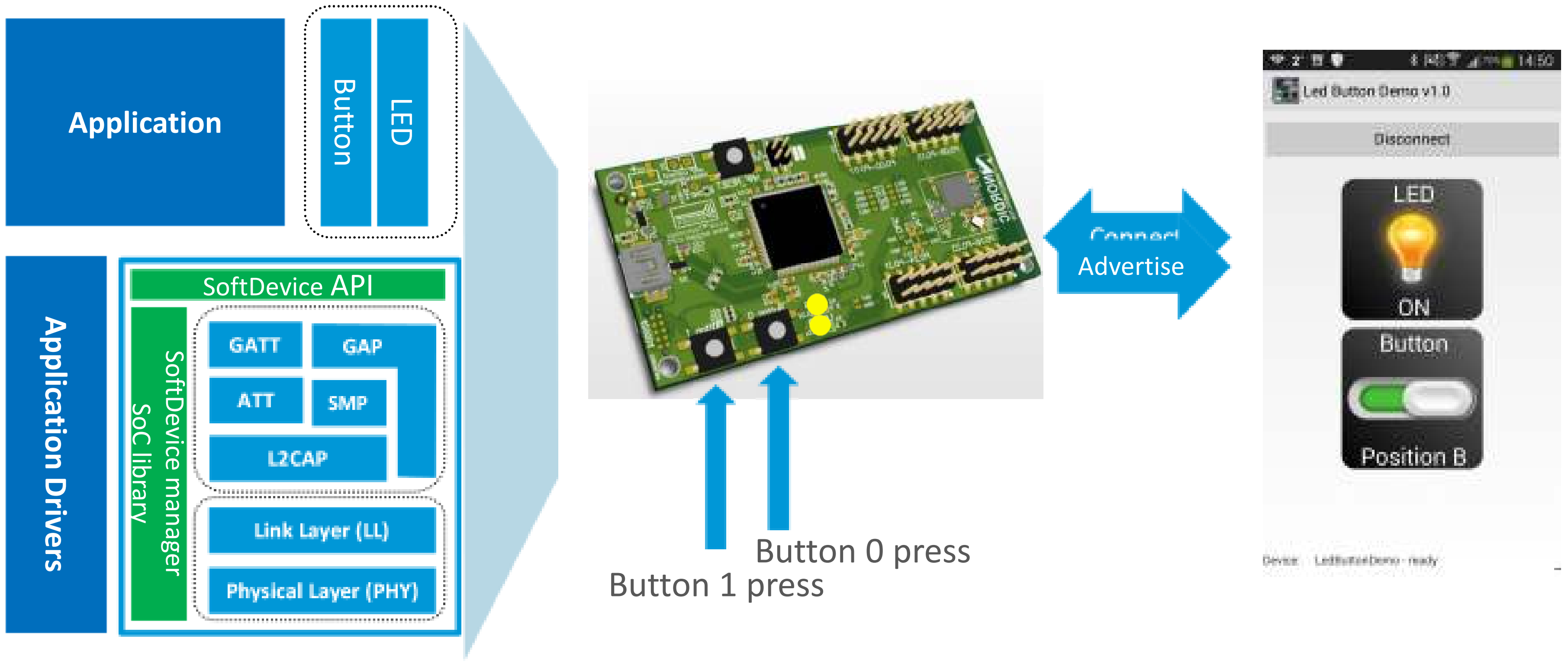
We will try to add the LBS service UUID into the advertising data. Advertising packet can contain 31 bytes

```
static void adver
{
        uint8_t flags
        ble_uuid_t ad

        // Build and set advertising packet data
        memset(&advdata, 0, sizeof(advdata));
        advdata.name_type = BLE_ADVDATA_FULL_NAME;
        advdata.include_appearance = true;
        advdata.flags.size = sizeof(flags);
        advdata.flags.p_data = &flags;

        // Build and set scan response packet data
        memset(&scanrsp, 0, sizeof(scanrsp));
        scanrsp.uuids_complete.uuid_cnt = sizeof(adv_uuids) / sizeof(adv_uuids[0]);
        scanrsp.uuids_complete.p_uuids = adv_uuids;
        err_code = ble_advdata_set(&advdata, &scanrsp);
}
```



```
Advertising Data
    CompleteLocalName: LedButtonDemo
    Appearance: 0x1234
    Flags: LimitedDiscoverable, BrEdrNotSupported
Scan Response Data
    ServicesCompleteListUuid128: 0x000015231212EFDE1523785FEABCD123
```

LED Button application



NORDIC SEMICONDUCTOR

NORDIC tech TOUR

Demo

NORDIC techTOUR

Building a GATT/GAP Profile

NORDIC SEMICONDUCTOR