# Wireless IoT Entwicklung mit Nordic Semiconductor

Kapitel 2: Bluetooth Peripheral LED & Button Service (LBS)

February 2026

nRF54L15DK / nRF Connect SDK 3.2.1

NORDIC® SEMICONDUCTOR

# Bluetooh Low Energy

## Hands-on LED/Button service (LBS)

© Nordic Semiconductor

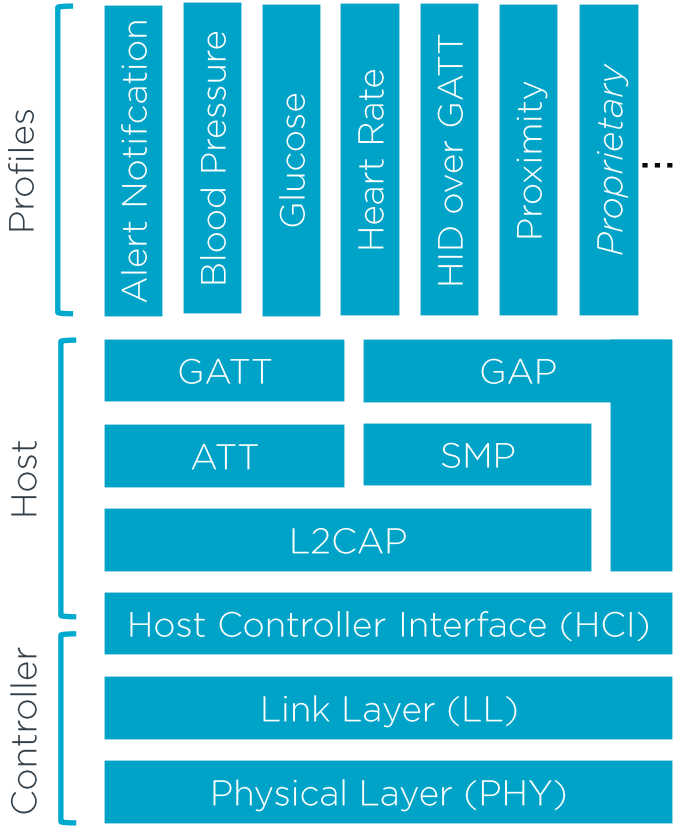
# Bluetooth SIG

- Bluetooth Special Interest Group
- Develop and license Bluetooth Low Energy technology
- Network of member organizations
- Founded in September 1998
- Non-profit
- 36000 member companies
- 5.4 billion Bluetooth product shipments in 2024
- Nordic is an associate member
  - Involved in several working groups
  - Help develop specifications

© Nordic Semiconductor

# Architecture

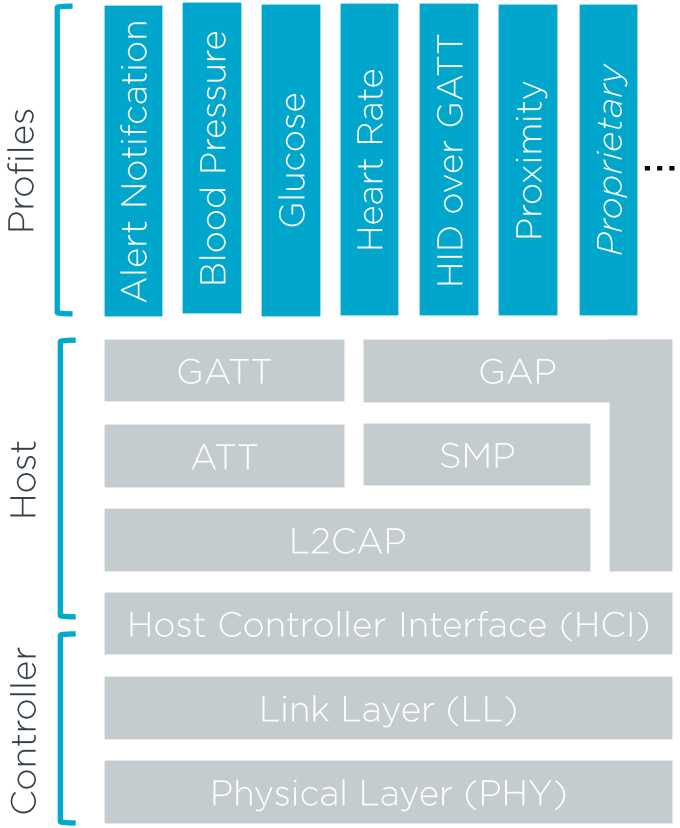
- Controller – Radio connection
- Host – Connection, discovery, advertising
- Profiles – Services and characteristics



© Nordic Semiconductor

# Profiles

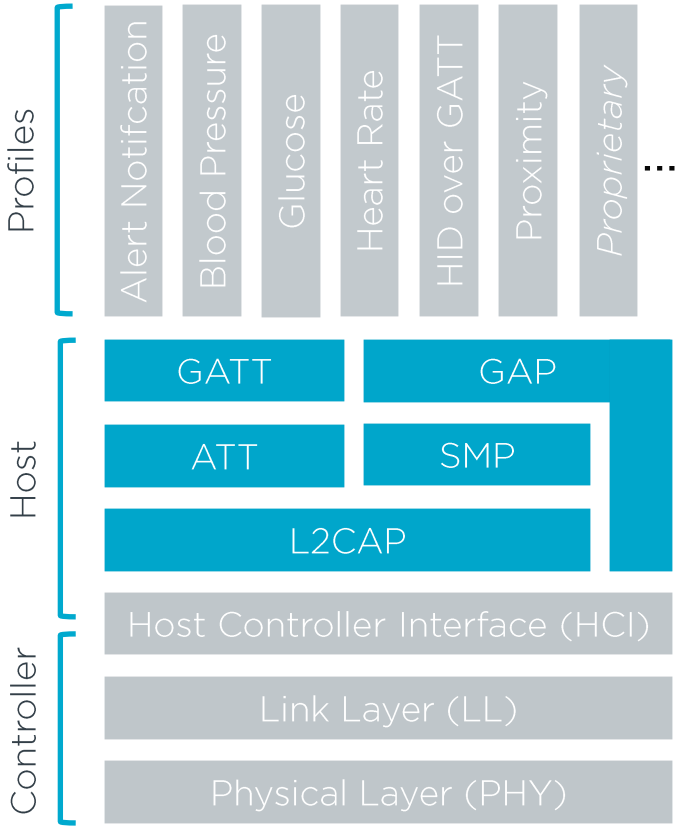
- Profiles
  - Describes how two or more devices can discover and communicate with each other
  - Implements a specific application
  - Standard or proprietary
  - Each profile has its own specification



© Nordic Semiconductor

# Host

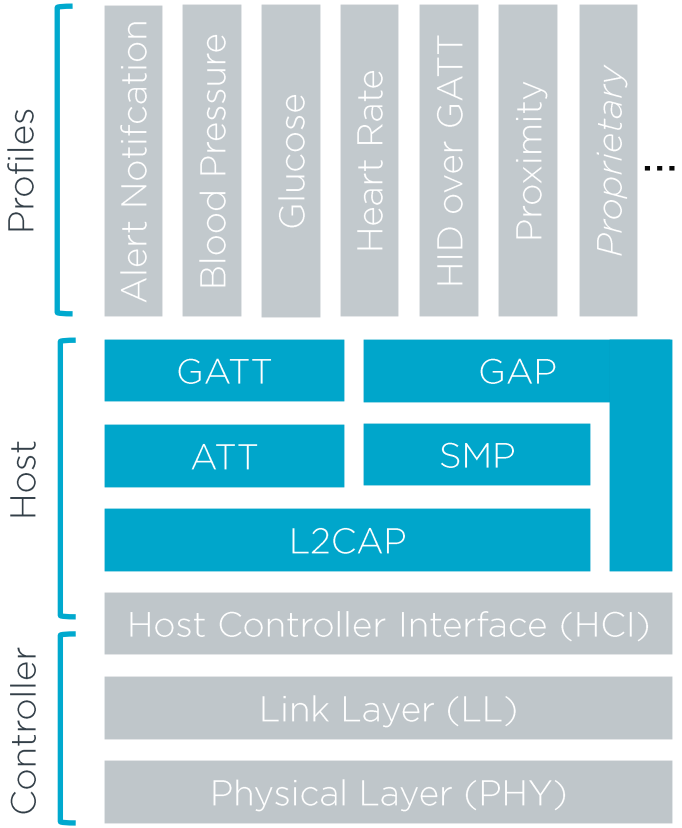
- Upper layers of the Bluetooth LE protocol stack
- Logical Link Control and Adaption Protocol
- Attribute Protocol (ATT)
  - Simple client-server model
  - Client device can access attributes on the server device
- Security manager Protocol (SMP)
  - Defines protocol for pairing and key distribution



© Nordic Semiconductor

# Host - GATT

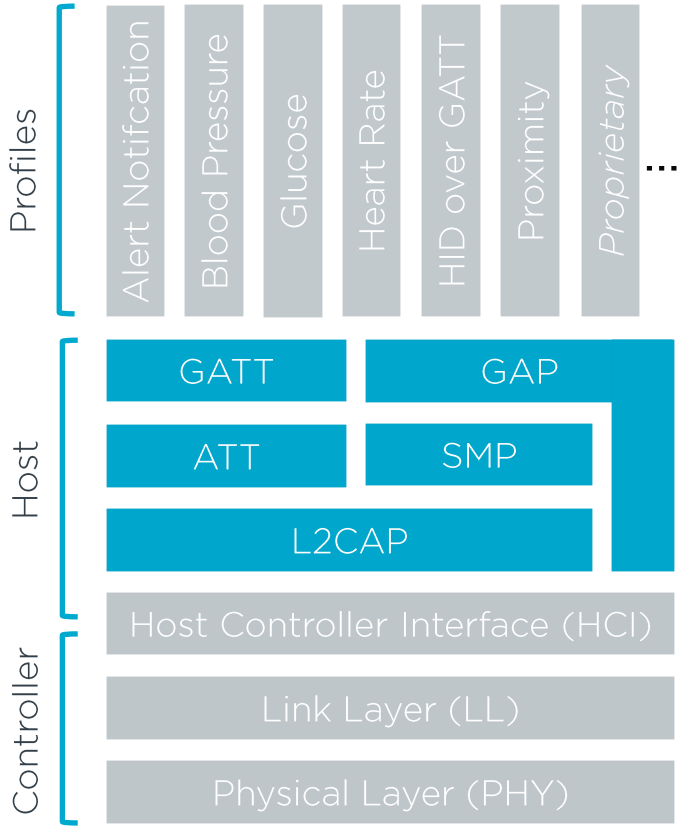
- Generic Attribute Profile (GATT)
- Highest data layer
- Uses ATT to discover and access attributes
- Specifies a hierarchical structure of attributes
  - Services
  - Characteristics
  - Descriptors



© Nordic Semiconductor

# Host - GAP

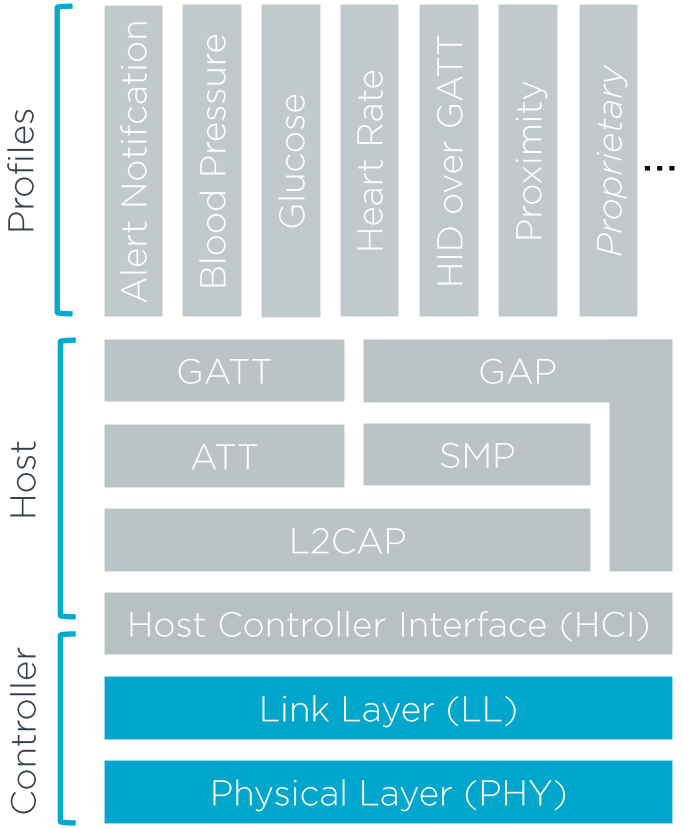
- Generic Access Profile (GAP)
  - Highest control layer
  - Defines device roles
  - Defines how devices discover and connect to each other
  - Defines security modes and procedures



© Nordic Semiconductor

# Controller

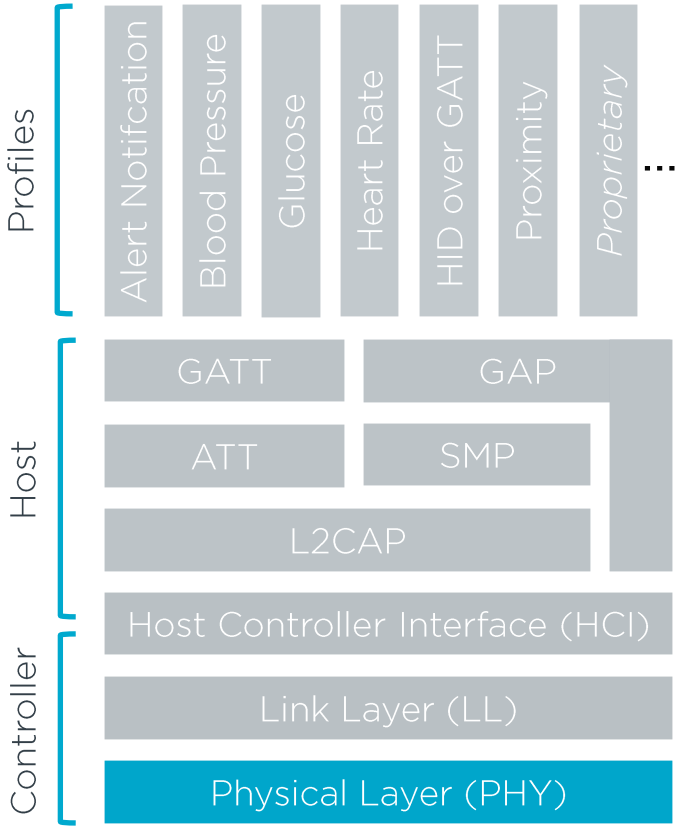
- Physical layer
  - Defines how two radios can send bits to each other
- Link Layer
  - Defines Link Layer states
  - Defines device address
  - Packet format



© Nordic Semiconductor

# Physical layer

- 2.4 GHz ISM band
- 40 RF channels (2 MHz)
- GFSK modulation
  - 1 Mbps or 2 Mbps
- Max. 20 dBm TX power
  - In Europe restricted to 10 dBm



© Nordic Semiconductor

# Physical layer – Radio Modes

- 1M PHY
  - Classis PHY Supported by all BLE devices (default mode)
- 2M PHY
  - Introduced in v5.0
  - Doubles the data rate, lowers power consumption.
  - Decreased sensitivity with translates to less range (about 80%)
- Coded PHY
  - Uses more symbols to send one bit
  - 2 different modes S2 500kbps (1Mbps/2) and S8 125kbps (1Mbps/8)
  - Longer transmission times resulting in longer range (about 4x)

© Nordic Semiconductor

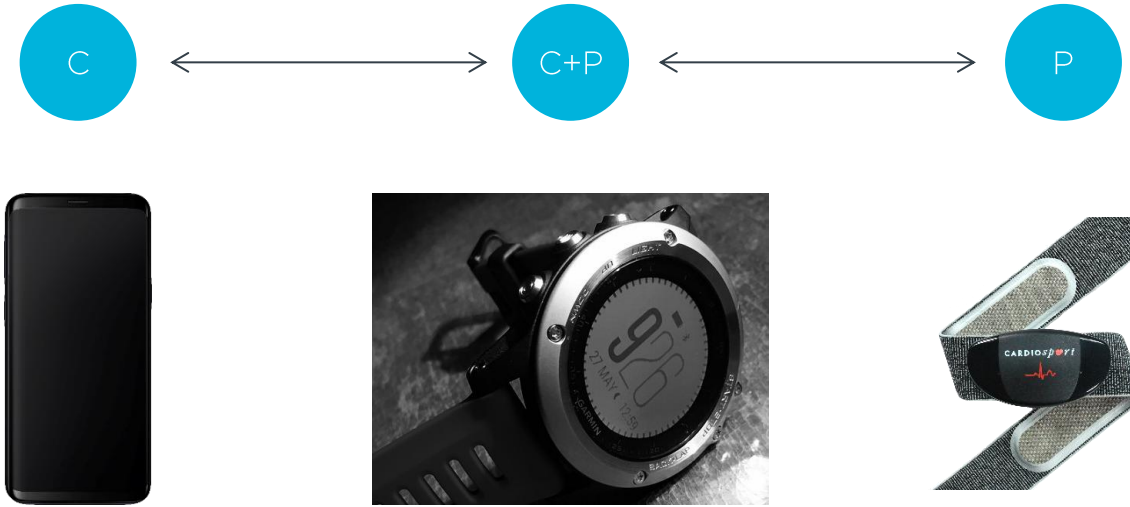
# GAP roles and Link Layer states

| GAP role | Link Layer state |
|---|---|
| Broadcaster | Advertising |
| Observer | Scanning |
| Peripheral | Advertising<br>Connection (Slave) |
| Central | Scanning<br>Initiating<br>Connection (Master) |

All roles can also be in the standby state

© Nordic Semiconductor

# Roles (GAP) - Example




B Broadcaster

O Observer

P Peripheral

C Central

# Let’s build it

## LED and Button Service (LBS)

© Nordic Semiconductor

# Nordic Led/Button Service

## The Hello World with Bluetooth Low Energy

- Build the Nordic “peripheral_lbs” sample
- Test application using the nRF Connect Mobile apps
- Change the BLE advertising type
- Add handler for connected state

Change the device name to unique name for broadcasting

Modify the name in prj.conf: `CONFIG_BT_DEVICE_NAME=“Your_Name”`

# Let’s test it

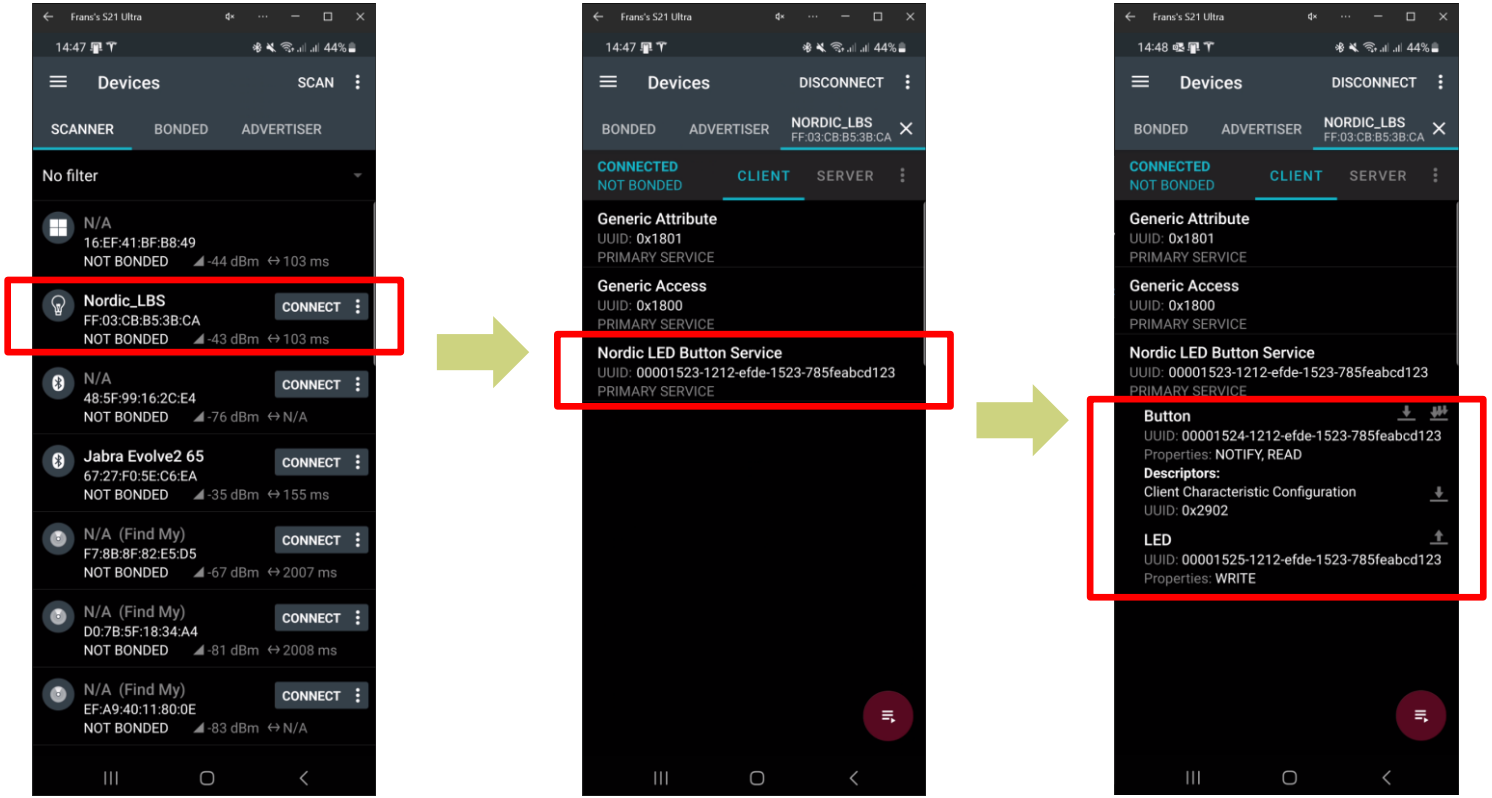
## LED and Button Service (LBS)

© Nordic Semiconductor

- LED 0:
  Blinks when the main loop is running (that is, the device is advertising) with a period of two seconds, duty cycle 50%.
- LED 1:
  Lit when the development kit is connected.
- LED 2:
  Can be activated/deactivated through a Bluetooth write operation.
- Button 0:
  Send a notification with the button state: "pressed" or "released".

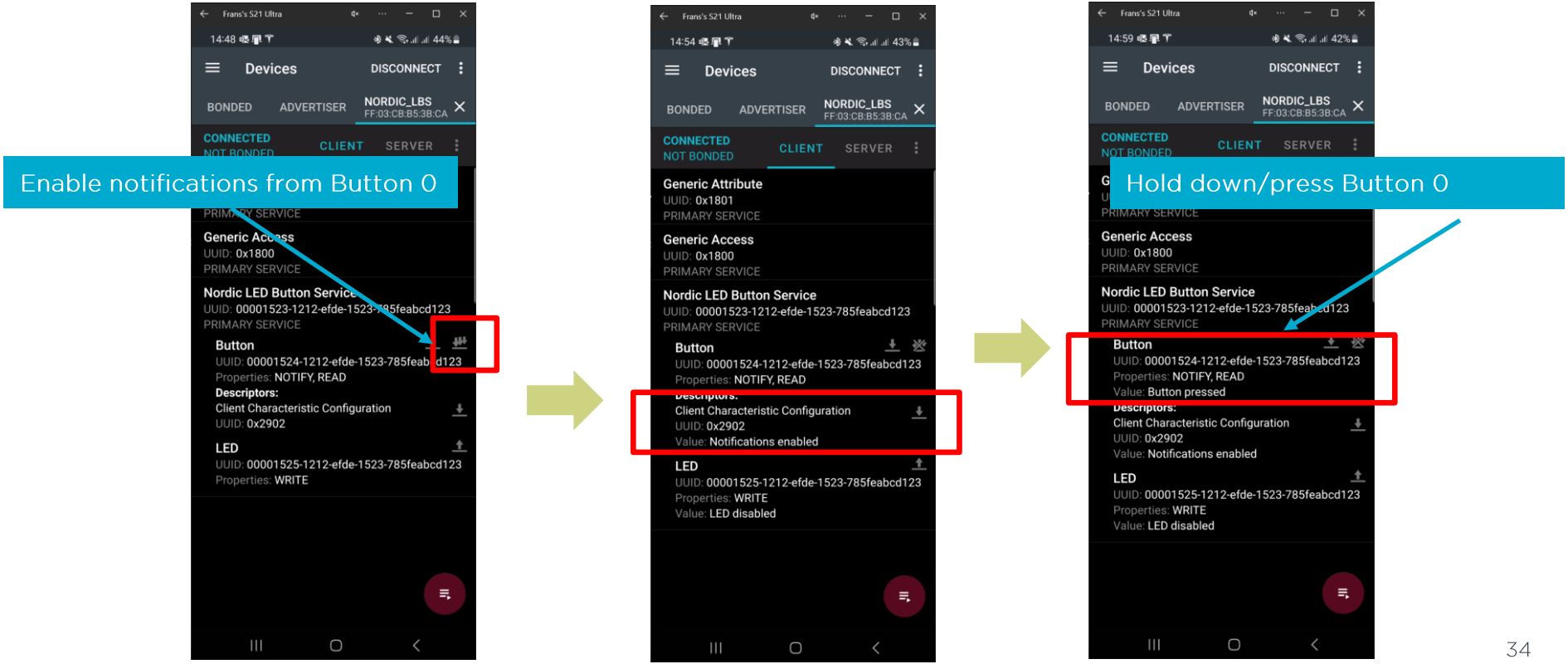
© Nordic Semiconductor

# nRF Connect for Mobile



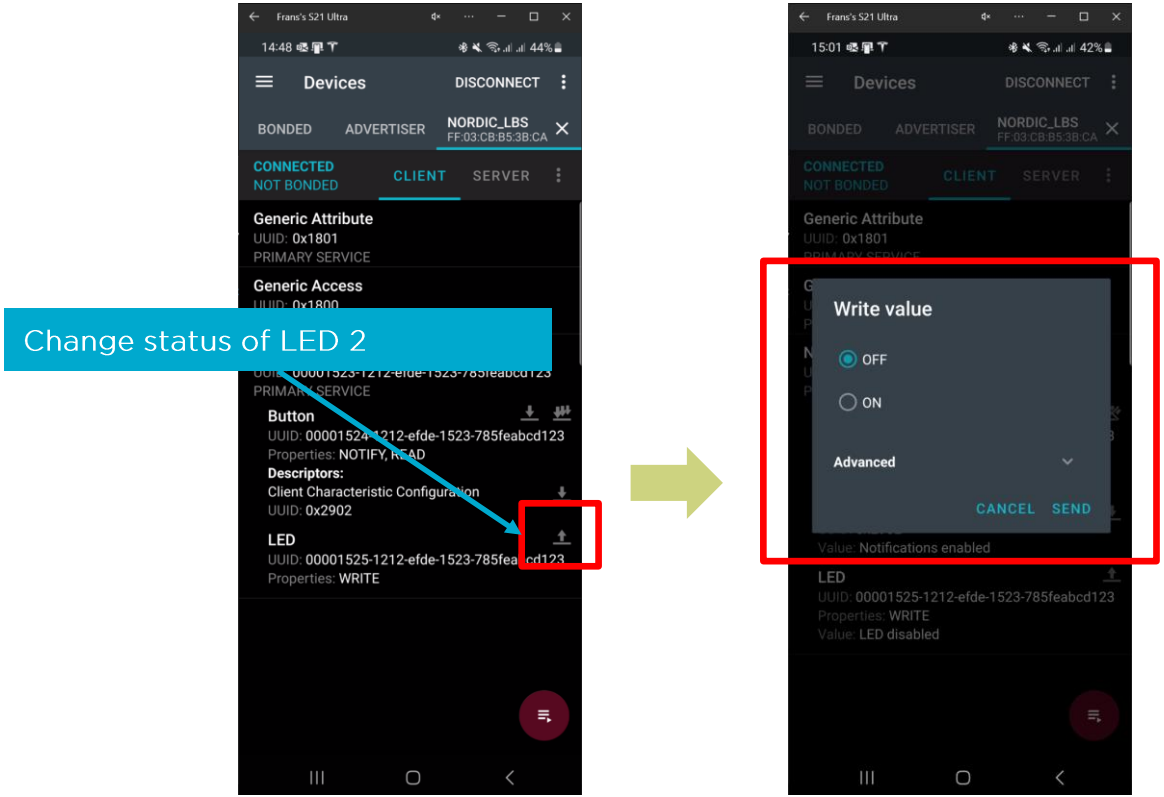
© Nordic Semiconductor

# View button state advertisement



© Nordic Semiconductor

# Change LED status on development kit




iOS users:

The Nordic LBS service, and specifically its LED state characteristic can be written with ByteArrays or Boolean values:

- 0x00 (off) / bool false
- 0x01 (on) / bool true

# Analyze the application

© Nordic Semiconductor

# Project configuration (file: prj.conf)

Enable Bluetooth

Setup as peripheral device

The name the devices will use during advertisement

```
CONFIG_BT=y
CONFIG_BT_PERIPHERAL=y
CONFIG_BT_DEVICE_NAME="Nordic_LBS"
```

Enable the Nordic Led Button Service

Enable the button feature

```
# Enable the LBS service
CONFIG_BT_LBS=y
CONFIG_BT_LBS_POLL_BUTTON=y
```

Enable the Development Kit library for adding support for the Leds and Buttons/switches

```
CONFIG_DK_LIBRARY=y
```

© Nordic Semiconductor

# Advertising Package

- Array with type struct bt_data which will be used to send out data. It consist of the advertising flags and the advertising packet data.

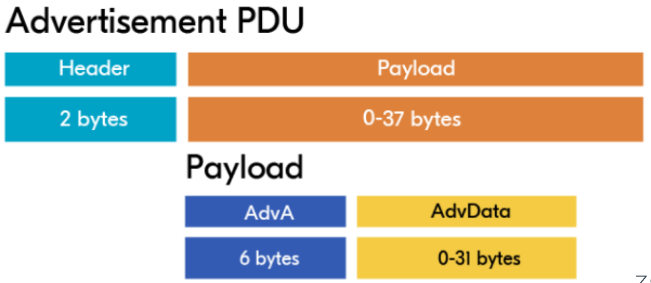
`BT_DATA_BYTES()` Is a macro to generate the header

- `BT_DATA_FLAGS` Indicator of package type
- `BT_LE_AD_GENERAL` Indicate that advertising is available for long time (no time out)
- `BT_LE_AD_NO_BREDR` Classic Bluetooth not supported

`BT_DATA()` Is a macro to generate the payload

- `BT_DATA_NAME_COMPLETE` Indicator of package type
- `DEVICE_NAME` Defined name in prj.conf
- `DEVICE_NAME_LEN` Length

```
43  static const struct bt_data ad[] = {
44      BT_DATA_BYTES(BT_DATA_FLAGS, (BT_LE_AD_GENERAL | BT_LE_AD_NO_BREDR)),
45      BT_DATA(BT_DATA_NAME_COMPLETE, DEVICE_NAME, DEVICE_NAME_LEN),
46  };
```



© Nordic Semiconductor

# Scan Response packet

- There are numerous standard data types that can be included in the advertising data (either in the advertising packet or the scan response). These data types are defined in the Bluetooth Supplement to the Core Specification.
- In this example we response with the UUID of the Nordic LBS service
- Like with the advertisement package the Macro BT_DATA_BYTES is used.

- `BT_DATA_UUID128_ALL` Indicate the data type (Standard BLE UUID)
- `BT_UUID_LBS_VAL` UUID for the Nordic LBS Service

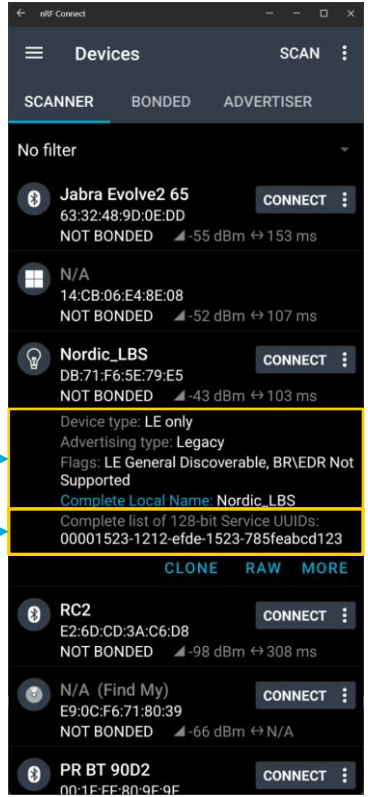
```
48  static const struct bt_data sd[] = {
49      BT_DATA_BYTES(BT_DATA_UUID128_ALL, BT_UUID_LBS_VAL),
50  };
51
```

© Nordic Semiconductor

# Advertising & Response (Service UUID)

Advertising information

Response package with Service UUID



© Nordic Semiconductor

# Define BLE call backs

Call back definition from HOST controller

*[connected, disconnected, security_changed,*

*le_param_req, le_param_updated, le_phy_updated, le_data_len_updated, identity_resolved, remote_info_available ]*

```
92  BT_CONN_CB_DEFINE(conn_callbacks) = {
93      .connected        = connected,
94      .disconnected     = disconnected,
95  #ifdef CONFIG_BT_LBS_SECURITY_ENABLED
96      .security_changed = security_changed,
97  #endif
98  };
```

Call back definition for LBS Profile

```
163 static struct bt_lbs_cb lbs_callbacs = {
164     .led_cb    = app_led_cb,
165     .button_cb = app_button_cb,
166 };
```

© Nordic Semiconductor

# Main function

Enable Bluetooth

```
223     err = bt_enable(NULL);
```

Load paired devices information from flash when enable

```
231     if (IS_ENABLED(CONFIG_SETTINGS)) {
232         settings_load();
233     }
```

Setup the callback programs for the BLE profile (led_cb & button_cb)

```
235     err = bt_lbs_init(&lbs_callbacs);
```

Start advertising using the define advertising package and scan response package. BT_LE_ADV_CONN defines advertisement type.

```
241     err = bt_le_adv_start(BT_LE_ADV_CONN, ad, ARRAY_SIZE(ad),
242                           sd, ARRAY_SIZE(sd));
```

© Nordic Semiconductor

# Buttons and LED Library

Button and Led library for development kits

```
27  #include <dk_buttons_and_leds.h>
```

Define LEDs for BLE connection status
Blink interval

Define LED which is controlled by user

Define which button is used to indicate status

```
33  #define RUN_STATUS_LED          DK_LED1
34  #define CON_STATUS_LED          DK_LED2
35  #define RUN_LED_BLINK_INTERVAL  1000
36
37  #define USER_LED                DK_LED3
38
39  #define USER_BUTTON             DK_BTN1_MSK
```

Call back function on button event

```
168 static void button_changed(uint32_t button_state, uint32_t has_change
```

Define call back for button events

```
182     err = dk_buttons_init(button_changed);
```

Init I/O pins used by LED library

```
197     err = dk_leds_init();
```

# Change Advertising

© Nordic Semiconductor

# Change Scan Response packet

- The content of the scan response packet can be chosen. Your free to choose what to show. Let change it to a website using a BT Uniform Resource Identifier (URI):

```
#define URL_STRING "//academy.nordicsemi.com"
static const struct {
           unsigned char prefix;
           const char url_string[sizeof(URL_STRING)];
} url_data = {
    .prefix = 0x17, // https
    .url_string = URL_STRING
};
static const struct bt_data sd[] = {
    BT_DATA(BT_DATA_URI, (const unsigned char *)&url_data, sizeof(url_data))
};
```

- `BT_DATA_URI` — Indicate the data type (URI)
- `url_data[]` — Text string with website / URL

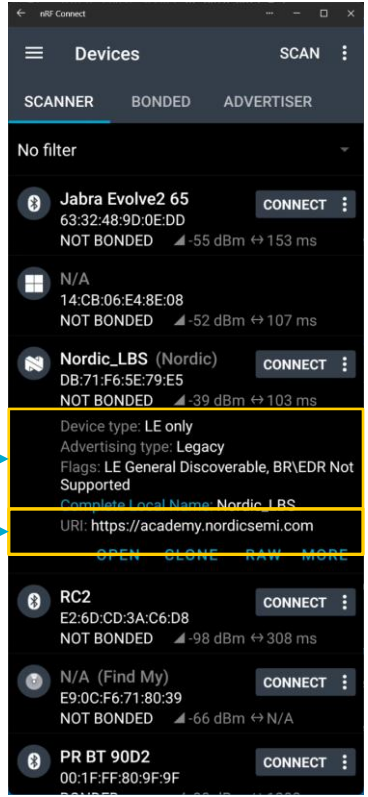
© Nordic Semiconductor

# Advertising & Response (Service URI)

Note: When removing the “Service UUID” the nRF Blinky mobile app will not recognize the development board any longer,

Advertising information

Response package with URL string (URI Data)




Note: The Uniform Resource Identifier (URI) is NOT visible on iOS

© Nordic Semiconductor

# Code challenge

LED0 stops blinking and is OFF when connected.

LED0 starts blinking when not connected and advertising

© Nordic Semiconductor

```
static bool app_connected_state;

static void connected(struct bt_conn *conn, uint8_t err)
   ....
   app_connected_state = true;

static void disconnected(struct bt_conn *conn, uint8_t reason)
   ....
   app_connected_state = false;

int main(void)
   ....
   for (;;) {
        if (app_connected_state) {
           dk_set_led_off(RUN_STATUS_LED);
        } else {
           dk_set_led(RUN_STATUS_LED, (++blink_status) % 2);
        }
        k_sleep(K_MSEC(RUN_LED_BLINK_INTERVAL));
    }
```

```c
static bool app_disconnected = true;

static void connected(struct bt_conn *conn, uint8_t err)
   ....
   dk_set_led_off(RUN_STATUS_LED);
   app_disconnected = false;

static void disconnected(struct bt_conn *conn, uint8_t reason)
   ....
   app_disconnected = true;

int main(void)
   ....
   for (;;) {
        if (app_disconnected) {
            dk_set_led(RUN_STATUS_LED, (++blink_status) % 2);
        }
        k_sleep(K_MSEC(RUN_LED_BLINK_INTERVAL));
   }
```

# Bluetooth LE Sniffer

DevAcademy – Lesson 6

© Nordic Semiconductor

# DevAcadamy - Lesson 6 – BLE sniffer

- Communication is real-time making data exchange debugging difficult.
- Setting break points will break communication.
- Lower layers are difficult to monitor from the application.

- Objectives
  - Learn how to use a Bluetooth sniffer to capture advertising packets
  - Learn how to install and run Nordic's Bluetooth sniffer, nRF Sniffer, to capture and analyze Bluetooth LE packets
  - Through nRF Sniffer, learn how to follow a connection and inspect the communication
  - Observe and decrypt the Bluetooth LE communication of a secured connection

- https://academy.nordicsemi.com/lessons/lesson-6-bluetooth-le-sniffer/