# Lab 2 - BLE Advertisements

**Goals**

- Send BLE advertisements as an advertiser
- Receive BLE advertisements as a scanner
- Encode/Decode BLE advertisement payloads

**Equipment**

- Computer with lab toolchain setup
- Three nRF52840DK and cables
- Smartphone iOS or Android (optional but highly useful)

## Lab Steps

### 1. Update your repository

- Pull in updates from https://github.com/brghena/nu-wirelessiot-base
  - cd into the base of your forked repo
  - `git pull https://github.com/brghena/nu-wirelessiot-base.git main`
  - Then merge and push to your own forked repo.

- Update the nrf52x-base submodule
  - cd into the base of your forked repo
  - `git submodule update --init --recursive`

### 2. Basic advertising application

There are now new apps that demonstrate advertising and scanning for BLE advertisements! We'll start off with the most basic of them: advertising a device name.

You'll need at least two nrf52840DK boards connected today, so might as well start off with both connected now. If you ever have them both advertise, make sure to modify their BLE addresses.

- cd software/apps/ble_adv_name/
- `make flash` to build and `make rtt` to view output
  - Note: You will have to choose a board by JTAG serial number
    - Feel free to write the number on the board with sharpie, on tape, on a sticky note, etc.
    - Or you can just remember the order in the selection pop-up. It is sorted and stable

## 3. Smartphone scanning (optional but highly recommended)

The easiest way to get a little introspection into what's going on is to use a smartphone to do so. Nordic Semiconductor actually released an app for Android and iOS that lets you test out a surprising amount of BLE behavior. On Android it can scan, connect, and advertise, with the ability to show you raw and interpreted details of observed packets. On iOS it may be more limited, but is still very helpful.

- Install the nRF Connect App: https://www.nordicsemi.com/Software-and-tools/Development-Tools/nRF-Connect-for-mobile

- You should now be able to do a scan and see the device that's advertising its name.
    - If you're in a house far enough away from neighbors and without Apple devices, it's possible you'll only see your device. Otherwise, you *probably* see a surprising amount of BLE devices, especially if you're living in an apartment. Take some time to see what you can figure out about them with the app and note how many there are.
    - Tip: You can use the filter to only show the ones you're interested in. I typically filter down to like -70 dBm RSSI (or higher), which is good enough to remove a lot of background devices. When your device and smartphone are within a meter, expect an RSSI of -30 to -50 dBm.

## 4. Scanning application

Next, let's get some introspection on a microcontroller. There is an app "ble_scan/" which has basic code for scanning for BLE advertisements. The function "ble_evt_adv_report()" gets called whenever a BLE advertisement is received. You'll need to write code to filter these advertisements for the device(s) you actually care about and to display details about their payloads.

This app will be useful when debugging things for the rest of the lab. So definitely keep it running in an RTT window and feel comfortable going back and modifying it so you can understand what's going on with other advertisement applications.

Remember that advertisement payloads are encoded as a series of length-type-value sets. The length field includes the number of bytes in the value plus 1 (for the type). The fields numbers are defined in the Assigned Numbers website. The value formats are defined in the Core Specification Supplement (link downloads a PDF) document, Part A "Data Types Specification".

Note: I found I had to keep unplugging/replugging my scanning board, because uploads would fail, and then when I tried again without unplugging everything would hang. After an unplug/replug everything would work and I could open RTT again. It was annoying though. Bugginess while trying to program is especially prevalent if you're printing a lot of information through RTT very quickly.

- Modify the scanner application to filter for devices you care about and print out useful information
    - Filtering is done with an if statement, usually based on parts of the BLE address
    - What qualifies as "useful" information is up to you
        - You could print raw bytes from payload
        - You could print field types and values
        - You could print some fields as strings or characters

## 5. Advanced advertising application

Some base code is available for sending raw advertisement payloads in “ble_adv_raw/”. It calls the simple_ble function simple_ble_adv_raw() (header documentation), which in turn calls sd_ble_gap_adv_set_configure().

Be aware that your input needs to be well-formatted! You’ll see the LED blink pattern like from the error app if it’s wrong, and RTT will print an error occurring from inside simple_ble.c.
Note: You may need to keep the Flags field in the advertisement for it to appear in scans from an iOS device.

- Modify the name of the device and add another field with some data.
    - You can choose any field you want.
        - TX Power Level and Manufacturer Specific Data are relatively straightforward options
    - A list of the type numbers is available online: https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile/
    - A specification for how to use that field is available in Core Specification Supplement v9 (link downloads a PDF)

## 6. Eddystone advertising application

Eddystone is one of several formats for encoding URLs into BLE advertisements. It encodes services and service data into the advertisement payload, and specifies how they can be decoded into a URL. The simple_ble library provides three helper functions, which can be used to transmit Eddystone packets:
https://github.com/lab11/nrf52x-base/blob/master/lib/simple_ble/simple_ble.h#L262

For the simple_ble functions, you provide a https URL skipping the “https://” part, which the simple BLE library automatically prefixes your URL with. So for example, if you want to direct people to “https://google.com” you provide the URL string “google.com”. The protocol (“https://”) is added to the packet as a shortened byte, 0x03. You can actually shorten the TLD as well, which would look something like “google\7”, but you aren’t required to. A full list of URL encoding bytes is listed here: https://github.com/google/eddystone/tree/master/eddystone-url

If you’ve done this correctly, and are using a smartphone to scan, the nRF Connect App should automatically decode the URL you transmit.

- Create a new application and have it transmit BLE advertisements with an Eddystone URL payload.
- Either with a smartphone or a scanning application, determine the type fields used in an Eddystone advertisement payload.
    - A list of the type numbers is available online: https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile/

## 7. Advanced scanning application

Finally, we'll parse an unknown BLE advertisement to determine what the fields are and what secret message it holds. The secret message app is available in "ble_adv_secret_message". I've provided a pre-built binary, which you can upload to a dev kit using the JLinkExe command listed in the README of that application. The BLE advertisements will be sent using the address c0:98:e5:4e:03:97.

Create your own scanner application that receives advertisements, filters for this device, parses the fields of the device, and finds and prints out the secret message.

- Load the secret message application onto a dev kit
- Create a new application that is a copy of your existing scanning application and load it onto another dev kit
- When parsing, be sure to check the format of each field in the CSSv9 document.
    - Hint: You may have to skip over irrelevant bytes to get the message properly
- When you find the message bytes, print them with the format string "%s\n". Be sure to give printf a pointer to the right location in the advertisement data.
    - Hint: the secret message might be encoded in unicode rather than ascii, but %s will handle it correctly.
    - Also, once you think you have things working, remove other repeating print statements from the application and change the format string to "%s\r" to get the full effect.

# Lab 2 Submission

1. Demonstrate that advertising and scanning worked. (screenshot here would do)

2. Approximately how many other devices can you see in a BLE scan? And (if you can tell) what types of devices are they?

3. Advanced advertising: Explain what additional field(s) you advertised and the overall byte layout of the payload.

4. Eddystone advertising: What are the type fields in an Eddystone URL packet?

5. Advanced scanning: What was the secret message?

6. Did you run into any particular issues?