***Internet of Things Course***

**LAB 4**

**A BLE Program**

**Issue 1.0**

Contents

Lab Overview 1

1 Implementation Details 2

1.1 Code reference 2

1.2 Setting up advertising 2

1.2.1 GAP attribute 2

1.2.2 GAP advertising parameters 2

1.2.3 Advertising modes 3

1.2.4 Advertising payload 3

2 Main Procedure Body 3

3 Profiles and Broadcasting 4

3.2 GATT profiles 4

3.3 Heartrate updates 4

4 Application Code 5

4.1 Program structure 5

## Lab Overview

In this lab we will learn how to program the DISCO-L475VG-IOT01A board, which has an on-board Bluetooth V4.1 module (SPBTLE-RF), and subsequently use a mobile device to connect to it via Bluetooth Low Energy (BLE).



Figure 1: DISCO-L475VG-IOT01A

In this lab we will program the embedded device and use it to simulate a heart rate sensor. In the following lab we will design an Android app to connect to this Bluetooth Smart heart rate sensor.

The embedded device will have all the same services and characteristics as a normal heart rate sensor, but instead of sending the Heart Rate (HR) measurements, the program will transmit dummy data. We will use the app we will develop later, to display the received data.

For the time being, you can debug the system using other apps that are available on app stores, e.g. nRF Connect

* for Android (<https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp&hl=en_GB>)
* for iOS (<https://apps.apple.com/gb/app/nrf-connect/id1054362403>)

# Implementation Details

## 1.1 Code reference

A detailed reference for the Mbed API’s BLE features can be found here:

<https://os.mbed.com/docs/mbed-os/v5.13/mbed-os-api-doxy/group__ble.html>

Note that we will be using the legacy advertising mode, as its features will be sufficient for our purposes.

## 1.2 Setting up advertising

To start, create a reference to a BLE instance in the global namespace of the program. The appropriate class, Instance(), is defined in the namespace BLE.

### 1.2.1 GAP attribute

Once you have initiated an instance of the BLE class, most of the program will proceed by making use of its attributes and their methods. The Generic Access Profile (GAP) attribute, ble.gap(), is initiated as a reference to an object of class Gap. It handles broadcasting (advertising), and connection for the device.

For example, once the advertising parameters and payload are set, ble.gap().startAdvertising(ble::LEGACY\_ADVERTISING\_HANDLE) will set the device to start broadcasting.

### 1.2.2 GAP advertising parameters

The details of how the device advertises (makes itself discoverable) are determined by setting the BLE advertising parameters. This is done by a call to ble.gap().setAdvertisingParameters, feeding in an advertising handle (it will suffice to use ble::LEGACY\_ADVERTISING\_HANDLE = 0x00), and an object in the class AdvertisingParameters.

This object will itself be constructed with two parameters: the advertising mode and a time interval, which will define what types of connections are allowed, and how often the device will advertise (trade-off between discoverability and power consumption), respectively. The former is described in more detail just below; the advertising-interval object can be constructed as:

ble::adv\_interval\_t(ble::millisecond\_t(ms)), where ms is a variable giving the time in milliseconds.

### **1.2.3 Advertising modes**

The other parameter to set is the GAP advertising mode, which is constructed with ble::advertising\_type\_t::type, where type is one of the AD types given below. Each will give the device the described connection properties:

AD types

* ADV\_NON\_CONNECTABLE\_UNDIRECTED – all connections to the peripheral device will be refused.
* ADV\_CONNECTABLE\_DIRECTED – only connections from a pre-defined central device will be accepted.
* ADV\_CONNECTABLE\_UNDIRECTED – any central device can connect to this peripheral.
* ADV\_SCANNABLE\_UNDIRECTED – any central device can connect to this peripheral, and the secondary Scan Response payload will be included or available to central devices.

There are other parameters whose values can be changed if desired – see documentation for details.

### 1.2.4 Advertising payload

As well as setting the advertising parameters, you will need to set the payload, i.e., the data being sent out by the device as it is advertising. To do this, we will use gap().setAdvertisingPayload, which again takes the advertising handle, but this time along with an object of type ble::AdvertisingDataBuilder. This can be constructed in one of two ways:

* either inline, with ble::AdvertisingDataSimpleBuilder, or
* as a named object with ble::AdvertisingDataBuilder.

For brevity, we will use the former, though you may wish to examine the latter, as it is easier to understand – it will require initiating an advertising buffer, which the simplified version bypasses.

Example code for the simple builder can be found in the documentation. In both versions, the payload is set by use of methods on the object, e.g., the appearance of the device can be set with .setAppearance(ble::adv\_data\_appearance::GENERIC\_HEART\_RATE\_SENSOR).

# Main Procedure Body

At this point we’ll pause from the BLE setup to write the rest of the procedure body, so that we can see the advertising setup being called, and test that the code so far is working.

The code that we use in the main procedure will represent some pretty complex workings “under the hood,” setting up an event queue and having the BLE device work through it. We will only need to work with this at a high level though; for instance, bleInitComplete, which should contain the advertising setup will be called automatically by the functions we will use in the main body.

ble.init(bleInitComplete) will execute this initialization. You’ll also want to link the BLE instance to the predefined event processing handler, using ble.onEventsToProcess. Finally, eventQueue.dispatch() will set the device going, and once you’ve loaded and run the program, you should be able to see the beacon “BLE Server” in whichever app you are using to discover devices.

# Profiles and Broadcasting

## 3.2 GATT profiles

Much like GAP being a module that defines how devices connect over BLE, GATT defines how devices share information. We will set up a GATT “profile” – a chosen set of services, which the BLE device will broadcast. Each of the services is a predefined collection of values called characteristics. For instance, a profile for receiving temperature from a digital thermometer could have services “temperature” and “device information,” the former having a single numerical characteristic and the latter having a couple of string characteristics. Services and characteristics are both identified by a 16 or 128 bit UUID.

To emulate what you might retrieve from a real heart rate monitor, we will have our device use the heart rate service as well as the battery service. All this takes is to initiate instances of HeartRateService and BatteryService, feeding in the BLE instance a value, and for HeartRateService a location (e.g., HeartRateService::LOCATION\_FINGER). With advertising correctly set-up, the battery service should already work to broadcast the input value.

## 3.3 Heart rate updates

To be able to see the heart rate service working properly, you will need to set up the value for periodic updates. This is fairly easy to do, by using the method eventQueue.call\_every. This takes a time period and the function that should be called periodically and adds it to the event queue appropriately.

# Application Code

This program demonstrates how to use the Generic Access Profile (GAP) for device advertising and connecting.

A device can send advertising GAP in two ways: the Advertising Data payload, which is mandatory for every device, and the Scan Response payload, which is optional. The Advertising Data payload is transmitted constantly and that is how peripheral devices inform a central device about their presence. The Scan Response payload is only transmitted when the central device asks for more information. Both payloads can contain only 31 bytes of data each.

The peripheral has a set interval that defines how often the device retransmits its advertising data. Shorter intervals will make the device more responsive but increase power consumption.

In this exercise, you will have to fill in the advertising data and configure parameters to make your board discoverable by other BLE devices.

## Program structure

Your program has to configure the system as a BLE device and update the Heart Rate Measurement characteristic. The program follows the structure below. Part of the functionality is already given.

* Initialization
  + Create BLE device, and DigitalOut objects.
  + Initialize variables and tables such as: device name, list\_of\_services, update\_characteristics\_flag, etc.
* Handlers
  + One Handler to toggle an LED to show activity; one Handler to raise a flag that indicates that the characteristics need to update.
  + One Handler to be called when the device is disconnected to restart advertising.
  + One Handler to indicate that the device has been connected via BLE.
* Main function
  + Start the BLE radio.
  + Setup the services.
  + Add the advertising parameter to the payload.
  + Start advertising.
  + If the flag is high, then update the heart rate measurement characteristic.
  + Wait for an event.