***Internet of Things Course***

**LAB 7**

**Weather Station**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc36672725)

[Lab overview 1](#_Toc36672726)

[2 Programming an Embedded BLE Weather Station 1](#_Toc36672727)

[2.1 Creating Customized Services and Characteristics 2](#_Toc36672728)

[2.2 Program Structure 3](#_Toc36672729)

[2.2.1 Mbed OS code 3](#_Toc36672730)

[2.2.2 Weather station app 3](#_Toc36672731)

[3 Exercise 6](#_Toc36672732)

# Introduction

## Lab overview

In this lab we will learn how to program the DISCO-L475VG-IOT01A board to operate as a weather station. This builds on previous labs where we learned how to send BLE advertisements, read from different sensors (including temperature), and connect to an embedded device via an app that we developed.

Here, the embedded device will use the health thermometer characteristic to broadcast temperature data taken from the on-board environmental sensors. We will extend the app that we have developed previously to display the received data.

Alternatively, you can debug the system using other apps that are available on the app stores, e.g., nRF Connect (<https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp&hl=en_GB>) or ST BLE Profile: (<https://play.google.com/store/apps/details?id=com.stm.bluetoothlevalidation>)

# Programming an Embedded BLE Weather Station

The starting point of this lab is the Mbed code we set up for a BLE “beacon” in Lab 4, as setting up advertising is most of the hard work. From this we will describe how to use Generic Attribute Profile (GATT) profiles to broadcast temperature data upon connection. The remaining work of retrieving the environment-specific measurements (temperature, humidity, etc.) from the sensors should be achievable by reusing parts of the code from the previous lab.

This lab exercise has two parts, namely programming the microcontroller device and programming the smartphone application. In this first part, we will program the board to interact with the on-board sensors using I2C serial communication (as we did in the “Sensing” lab), and then broadcast the data via the on-board BLE module.

In the second part, we will receive and display the data (temperature, humidity, atmospheric pressure, and cardinal direction) on the mobile device. Note that the cardinal direction can be used to indicate the direction of the wind if we can attach the device to a wind vane (or a weathercock).

In order to make our device compatible with other BLE apps, we will use the standard Universally Unique Identifiers (UUIDs) for the BLE services and characteristics. More information on the standard UUIDs can be found at: <https://developer.bluetooth.org>

The table below describes the details of the Environmental Sensing Service and the characteristics relevant to our application.

Table 1: Environmental Sensing Service Charecteristics

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **UUID** | **Format** | **Description** |
| Environmental Sensing (Service) | 0x181A | - | The Environmental Sensing Service (ESS) shows measurement data from an environmental sensor. |
| Humidity (Characteristic) | 0x2A6F | uint16 | Unit is in percentage, with a resolution of 0.01 percent. |
| Temperature (Characteristic) | 0x2A6E | sint16 | Unit is in degrees Celsius, with a resolution of 0.01 degrees. |
| Pressure (Characteristic) | 0x2A6D | uint32 | Unit is in Pascals, with a resolution of 0.1Pa. |
| True Wind Direction (Characteristic) | 0x2A71 | uint16 | Direction from which the wind blows. Angle measured clockwise relative to Geographic North. Unit is in degrees, with a resolution of 0.01 degrees. |

## Creating Customized Services and Characteristics

In the previous heart rate monitor program, we used predefined services that are part of the BLE\_API. In this exercise, we will create our own customized services and characteristics.

* We first need to define the service and its functions in a separate header file, as in the MBED\_API. We recommend using the EnvironmentalService.h file, since it provides a template and a fairly simple notification service. The file can be found in the project folder >mbed-os >features->FEATURE-BLE->ble-> services. We also need to declare our BLE device and characteristics as GattCharacteristic objects.

private:

BLEDevice &ble;

GattCharacteristic my\_characteristic;

* Then, let’s create the service constructor and pass it to our characteristics and its attributes. These include UUID, first value, size of the packet in bytes, size of the whole message in bytes, and notification properties.

my\_characteristic (GattCharacteristic::UUID\_HUMIDITY\_CHAR,&humidity,

GattCharacteristic::BLE\_GATT\_CHAR\_PROPERTIES\_NOTIFY),

* After that, we need to add all the characteristics to a GattCharacteristic table and declare the GattService with parameters including UUID (in this case we will use Environmental Service), the table of characteristics, and the number of characteristics.

GattCharacteristic \*charTable[] = {&my\_characteristic1, &my\_characteristic2};

GattService     my\_service(GattService::UUID\_ENVIRONMENTAL\_SERVICE,

charTable, sizeof(charTable) / sizeof(GattCharacteristic \*));

* Finally, we need to add our service to the Bluetooth Low Energy device object.

ble.gattServer().addService(my\_service);

* We also need to write some methods to interact with those characteristics easily. Each notification characteristic needs only one function to update the value. We use function updateCharacteristicValue to send the new value to the client.

ble.gattServer().write(myCharecteristic.getValueHandle(), (uint8\_t \*) &variable, size of the variable);

## Program Structure

### Mbed OS code

The program has to configure the system as a discoverable BLE device in the same way we did for the heart rate monitor exercise. The program structure will be the same; however, the services and characteristics broadcasted by the device will be different.

The current release version of the BLE\_API only allows us to have two notification characteristics in each service. Therefore, to be able to subscribe all the four parameters, we can create a custom Weather Station Service that will subscribe to characteristics like humidity, temperature, pressure, and wind direction. We will also add the Device Information Service, and optionally the Battery Service. The structure will be as follows. We can use a BLE analyzer app to check if the program structure is correct.

Figure 1: Weather Station Service structure

### Weather station app

In the second part of this lab exercise, we will design and build an app to connect to our weather station using Bluetooth Low Energy, and display the information retrieved on the screen.

The app will be similar to the one we created for the heart rate monitor; however, this time we will subscribe to four notifications. In addition, we will see how to add images to make our app more attractive.

* **User Interface**

The user interface is very simple as seen in Figure 2. At the bottom, we have the Connect button. Similar to the heart rate monitor application, when the button is pressed, our smartphone scans for Bluetooth Smart devices in range and displays a selection dialog with the devices found.

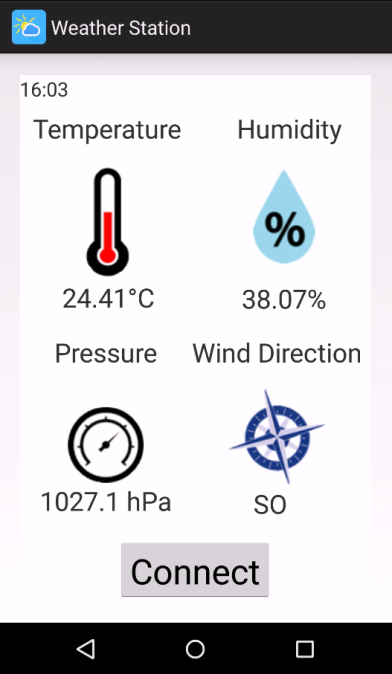
The rest of the screen is divided into four parts, one for each measurement. In each part, we have a title, which is a TextView object in Android Studio; an image representing the measurement, which is an ImageView object; and another TextView object to display the measurements.

Figure : Weather Station UI

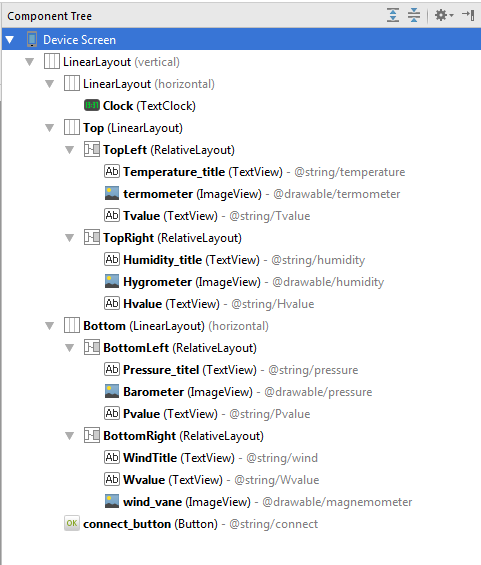


Figure : Screen Layout in Android Studio

To keep the objects at the correct position regardless of the size of the screen or the orientation, Android Studio provides us with Layout objects. There are different types of layouts that you can use according to your needs or preferences. For instance, a Horizontal Linear Layout will arrange all the objects on it in a single row, one after the other. A Vertical Linear Layout will arrange them in a single column. Alternatively, a Relative Layout arranges each object in a position relative to the Layout of the objects around it. You can use layout objects one inside another to achieve the desired result. See the above Component Tree as an example.

Then, to show the text or the images, we first have to add them into the resources folder in our project. For the text, we need to define some strings inside the file strings.xml. We can find this file in the project structure, under app > res > values. Edit the Text properties on the TextView object to link the text. For the images, we have to copy the files into the folder app > res > drawable. We can drag and drop the files. Then, edit the src properties to link the image to the object.

* **Main Code Structure**

The main activity program for this app will have a similar structure to that of the previous heart rate monitor app we created. We will use the same code to discover the devices in range when the Connect button is pressed. Then, the program will list them and establish a connection with the selected one, as we did previously.

Then, it is our task to complete the callback functions onServicesDiscovered and onCharacteristicChanged.

* **On Services Discovered Callback**

OnServicesDiscovered Callback is called after a client is paired with a master and starts discovering its services. In this callback, the program needs to check all the services and characteristics of the device and subscribe to the notifications that are supported by the app. There are different ways to do so. Below is one of the approaches:

First, we can make a list with all the Services. Following that, we can get the characteristics of each service and add them into a list of characteristics.

List<BluetoothGattService> gattServices = getSupportedGattServices();

for (BluetoothGattService gattService : gattServices) {  
List<BluetoothGattCharacteristic>gattCharacteristics=gattService.getCharacteristics();}

Once we have the list of characteristics, we can check if any of them matches with those supported by your app.

if (gattCharacteristic.getUuid().equals(AssignedNumber.getBleUuid("Humidity"))) {

If so, then the program should subscribe to the notification. First, we use the BluetoothGatt function setCharacteristicNotification to enable notifications for the given characteristic.

mGatt.setCharacteristicNotification(gattCharacteristic, true);

With this, our smartphone will accept notifications. However, this is not enough. We also have to set the BLE device to send notifications, by editing its Client Characteristic Configuration.

The recommended way to do so is using the BluetoothGattDescriptor class. The GATT Descriptors contain additional information and attributes of a GATT characteristic. They can be used to describe the characteristic’s features or to control certain behaviors of the characteristic. More details about this class are available at <https://developer.android.com>.

We first have to create a new BluetoothGattDescriptor linked with the Client Characteristic Configuration. Then set the value descriptor ENABLE\_NOTIFICATION\_VALUE. This value is used to enable notifications for a client configuration descriptor. Finally, we need to write the GATT descriptor. At the bottom of the activity file, the methods writeGattDescriptor and callback onDescriptorWrite have been defined to help you with the subscription task. These functions deal with the descriptorWriteQueue.

BluetoothGattDescriptor descriptor = gattCharacteristic.getDescriptor

(AssignedNumber.getBleUuid("Client Characteristic Configuration"));

descriptor.setValue( BluetoothGattDescriptor.ENABLE\_NOTIFICATION\_VALUE );

writeGattDescriptor(descriptor);

* **On characteristic changed Callback**

OnCharacteristicChanged Callback is called when a characteristic value is changed. In this callback, the program needs to check which characteristic has changed, get the value, convert the value to an appropriate format, and update the TextView.

To do so, we first need to compare the UUID number that has triggered the callback with the UUID of the supported characteristics. Once we know which characteristic has changed, we will use the function getIntValue to read the value from the device.

float humidity100 = characteristic.getIntValue

(BluetoothGattCharacteristic.FORMAT\_UINT16,0).floatValue();

final float humidity = humidity100 / 100.0f; // 2 decimals

Then, to update the user interface on the UI thread, we need to use runOnUiThread(). The action will be posted to the event queue of the UI thread.

Inside the runnable, define a TextView object linked to the string value that needs to be updated, and then use the functions setText and String.format to update the text.

TextView humidityTxt = (TextView) WeatherActivity.this.findViewById(R.id.Hvalue);  
humidityTxt.setText(String.format("%.2f%%", humidity));

# Exercise

You have to create an app to interact with the embedded weather station via Bluetooth Low Energy. You should create an easy-to-use user interface that displays the environmental data. Follow the applicable Bluetooth specifications so that the same program can interact with any BLE device.