***Internet of Things Course***

**LAB 8**

**WiFi Lab**

**Issue 1.0**

**Contents**

[1 Introduction 3](#_Toc38459331)

[1.1 Lab overview 3](#_Toc38459332)

[2 Device Management Platform 3](#_Toc38459333)

[2.1 Connecting devices 3](#_Toc38459334)

[2.2 Managing devices 4](#_Toc38459335)

[2.3 Updating device firmware 5](#_Toc38459336)

[3 Implementation 6](#_Toc38459337)

[3.1 Setting up the device management platform 6](#_Toc38459338)

[3.2 Setting up board 6](#_Toc38459339)

[3.2.1 Setting up WiFi 6](#_Toc38459340)

[3.2.2 Creating M2M resources 7](#_Toc38459341)

[3.2.3 Adding API key and connect certificate 7](#_Toc38459342)

[4 Application Code 7](#_Toc38459343)

[4.1 Program structure 7](#_Toc38459344)

# Introduction

## Lab overview

In this lab we will learn how to program the DISCO-L475VG-IOT01A board that has an on-board WiFi module to send and receive sensor data from the device management platform. This lab consists of two parts; the first part includes setting up the board to send sensor values using WiFi and the second part includes setting up the device management platform to view the readings of sensor from our board.

# Device Management Platform

The device management platform is a managed service for interacting and managing your connected devices. It includes:

* A secure connection, over which we can interact and manage our IoT devices.
* A dashboard that gives summary of our account like devices’ status and usage.
* Access Management for creating API keys to grant access to the device management portal.
* An end-to-end remote firmware update solution, including support for delta updates suitable for low-bandwidth and mesh-based networks.
* A flexible provisioning mechanism that can work with any factory setup to give our device all the information it needs to connect to device management.
* Web application development using a choice of interfaces: Representational State Transfer (REST) APIs or online portal.

## Connecting devices

Device management provides a secure connectivity solution between constrained IoT devices and software or web apps. In this section we will learn some key concepts on how the board is bootstrapped to the device management services and learn about lightweight machine-to-machine (LwM2M) protocol to communicate with your board.Following are some key aspects involved in the connection flow:

* **Device ID:** It is a unique ID generated by device management when the device connects for the first time. APIs can use this ID to refer to the device.
* **Endpoint name:** The name that is given to our device in the factory. This usually is the same as the device ID
* **Bootstrapping:** The device first connects to the bootstrap service, which provides the LwM2M server account credentials for registration.
* **Registration:** In this process the device directly connects to the LwM2M server using the credentials received during bootstrapping.
* **Deregistration:** A process of deregistering a device from the LwM2M server when it no longer needs connectivity.

## Managing Devices

The device management platform provides a way to interact with deployed devices. Devices have several built-in attributes: information that the device provides to device management, such as manufacturer, serial number, and model. Custom attributes allow us to create fine-grained filters. We can filter devices by both built-in and custom attributes.

* **Resources**

Resources are sensors and actuators in a device. Device resource management means, for example, reading a temperature value or controlling a device-activated door lock. You can use the device management services to manage device resources and execute actions on the devices. Resource management is based on read, write, create, and execute operations. All communication to the device is asynchronous and the result of an operation is delivered as an event through the event notification channel. To ensure the delivery, device management queues the operations and provides protocol agnostic retry logic.

Resource model has a hierarchical data structure representing the device information using **Objects** and **Resources**.

The device application creates and owns the resource structure:

* **Object:** An Object is a collection of Resources in the device. It is an “interface” to a temperature sensor, for example.
* **Object Instance:** An Object can have multiple Instances.
* **Resource:** Any piece of information from a device is a Resource, which has either a **static** value (a value that never changes) or a **dynamic** value (a value that changes during the device’s operation). You can read, write to, or execute a Resource’s information. A Resource can also be an array with multiple values.

**Sample Data Structure:**

Figure 1 shows an example of data structure with two different objects: a temperature sensor and a humidity sensor. The temperature sensor Object has five different Resources. We can read their values, reset their min/max values, and change the unit value. The device has three instances of a temperature sensor.

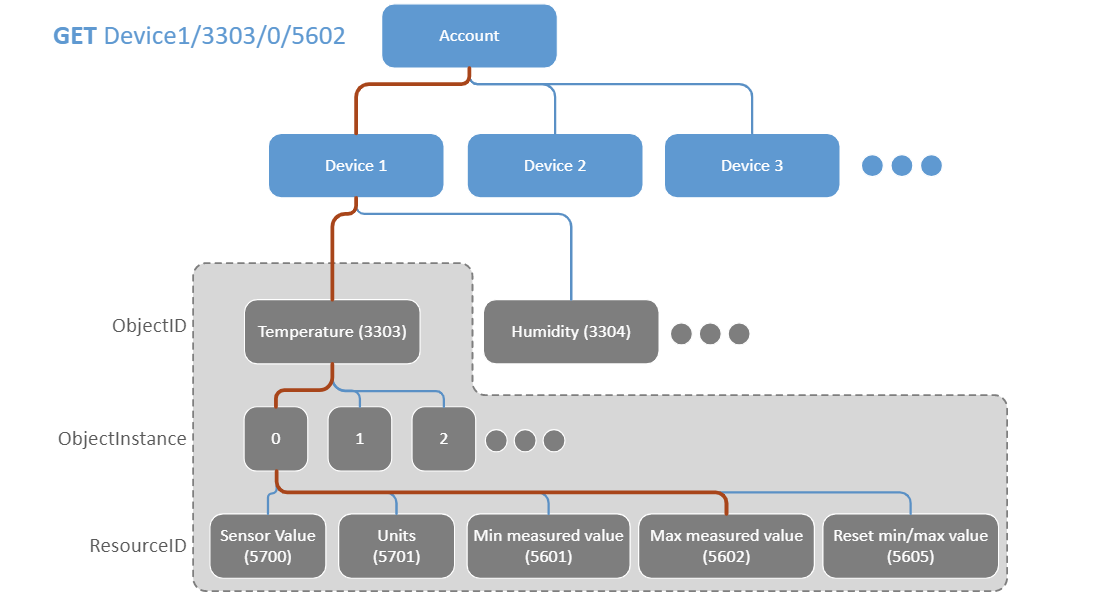


Figure : Sample Data Structure

## Updating device firmware

Device Management Update can be used to send new firmware to deployed devices. To do this, we need the device management Update *client* running on the device and the device management Update *service* delivering the updates. The process of deploying a firmware update to devices in the field begins with building a new firmware image and uploading it to the device management servers, where the update service can access it. The Update client, running on the device, is responsible for verifying, installing, and reporting the progress of firmware updates. We need to integrate it with your application to enable future updates. We can upload firmware to device management using:

1. The device management Portal.
2. The device management APIs.
3. The manifest tool update command.

Device Management Update uses certificates to ensure end-to-end security and to validate that a firmware update originated from a genuine and trusted source. It uses *elliptic curve cryptography* (ECC), a type of public key cryptography. In particular, the manifest tool and device management Update use the Elliptic Curve Digital Signature Algorithm. This algorithm uses two pieces of information:

* A secret, called the **private key**.
* A number, called the **public key**, which corresponds to the secret.

To sign a document, the holder of the private key first computes a hash of the manifest using SHA256 and then transforms that hash using the private key. This new number, called the **signature**, is appended to the document. Later, anyone with the public key can verify that the signature matches the document by:

* Computing a hash of the document using SHA256.
* Transforming the signature back into a hash using the public key.
* Comparing the computed hash and the signature hash.

If they are the same, then the verifier can be confident that the signer was in possession of the private key and therefore can validate the firmware update process.

# Implementation

## Setting up the device management platform

In this lab we will use Pelion device management platform. To use the device management services we just need to sign up using our existing Mbed account at <https://portal.mbedcloud.com/>

Once the account is set-up, we can create API keys under Access Management. API keys allow applications (mobile, web, and so on) to access the device management service APIs for a specific team and its devices. To create a new API key:

1. Click **New API key** in **Access Management** > **API keys**,
2. Give the API key an easily recognizable name.
3. Give access permission to the API key. By default, an API key uses the Developer group’s access permission. We can associate it with a different group either when creating it or later.
4. Click **Create API key**.
5. Copy the API key with the **Copy to Clipboard** button

## Setting up board

In the lab 5 we learnt how to read data from the sensors on-board. In this lab we will send the data from those sensors to the device management platform using WiFi.

### Setting up WiFi

Mbed OS boards should have a default configuration for connectivity and storage in the Mbed OS (targets.json). We can extend or override the default configuration using mbed\_app.json in our application by creating a new entry under the target name for our device where we set our WiFi credentials.

We initially set the default network instance,

NetworkInterface::get\_default\_instance();

Then use *connect()* function to connect to WiFi.

### Creating M2M resources

We need to create resources that will update the sensor values periodically. Below example shows how to send data to the device management system:

M2MObjectList m2m\_obj\_list;

// GET resource 3000/0/5701

m2m\_get\_res = M2MInterfaceFactory::create\_resource(m2m\_obj\_list, 3000, 0, 5701, M2MResourceInstance::INTEGER, M2MBase::GET\_ALLOWED);

cloud\_client->add\_objects(m2m\_obj\_list);

m2m\_get\_res->set\_value();

This code snippet binds a resource (any value) with a web path /3000/0/5701. The path can be specified randomly. Any update of the resource can be pushed to the device management platform through M2M protocols, by calling the function *set\_value(),* and you can view the updates simultaneously on the portal.

### Adding API key and connect certificate

We need to add the API key that we created in the device management platform. Next, we will add a connect certificate that will allow our device to securely connect to the device management system.

# Application Code

In this lab exercise we will write a program that reads the temperature and humidity values from the sensors on-board every 5 seconds. We will then send these values to the device management platform.

## Program structure

* Initialization
  + Create a DigitalOut object for the LED.
  + Initialize variables.
* Handlers
* Update sensor readings.
* Raise a flag that indicates that the measurements need to be read and displayed again.
* Main function
* Initialize sensors.
* Initialize WiFi.
* Create resources and send values.