Zurich University
of Applied Sciences

School of
Engineering
InES Institute of
Embedded Systems

# IoT Laboratory 5 –
# OMA lightweight M2M (LWM2M)

## Agenda

This is a shortened laboratory and is planned to last no longer than 90 minutes. The rest of the time you can use to work on your own IoT project.

The walkthrough should provide you an overview of OMA lightweight M2M (LWM2M). As example implementation we use Leshan, a project powered by the Eclipse Foundation. It is an OMA lightweight M2M (LWM2M) implementation in Java and allows you to manage devices (things) and their resources.

## Exercise Grading

3 points maximum will be awarded for completion of the exercises.

## Prerequisites

1. Conventional laptop
    - Python 3 interpreter installed
        - The code template uses the libraries:
          (If one of those is missing on your machine, install it via a Python package manager like pip)
            - json
            - urllib
            - codecs

Your laptop / development machine must reside in the laboratory subnet, 172.16.32.0/24. Preferably with all other network interfaces disabled (WLAN, …).

If you do not want to have a Python interpreter installed natively on your machine, use a Linux virtual machine or your Raspberry Pi device which you used in the previous laboratories.

# Table of Contents

**Agenda** ..... **1**

**Exercise Grading** ..... **1**

**Prerequisites** ..... **1**

**Getting started** ..... **3**

OMA (Open Media Aliance) ..... 3

Goals of OMA ..... 3
OMA LightweightM2M v1.0 ..... 4

Leshan ..... 4

**Scenario of this laboratory** ..... **4**

Procedure ..... 5

Available IoT devices in TE523 ..... 6
Naming authority ..... 6

**Implementation of a simple HTTP client** ..... **8**

Testing the access to the IoT devices ..... 8

Implementing an automated client ..... 8

Code examples ..... 9

**Capture LWM2M / CoAP traffic** ..... **10**

Setup / Scenario ..... 10

# Getting started

Primarily a few background information:

## OMA (Open Media Aliance)

**OMA is the Leading Industry Forum for Developing Market Driven – Interoperable Mobile Service Enablers.**

OMA is a non-profit organization that delivers open specifications for creating interoperable services that work across all geographical boundaries, on any bearer network. OMA's specifications support the billions of new and existing fixed and mobile terminals across a variety of mobile networks, including traditional cellular operator networks and emerging networks supporting machine-to-machine device communication.

OMA is the focal point for the development of mobile service enabler specifications, which support the creation of interoperable end-to-end mobile services. OMA drives service enabler architectures and open enabler interfaces that are independent of the underlying wireless platforms. Toward that end, OMA has developed programs that allow implementers the opportunity to test their products to ensure industry-wide interoperability.

### Goals of OMA

- Deliver high quality, open technical specifications based upon market requirements that drive modularity, extensibility, and consistency amongst enablers to reduce industry implementation efforts.
- Ensure OMA service enabler specifications provide interoperability across different devices, geographies, service providers, operators, and networks; facilitate interoperability of the resulting product implementations.
- Be the catalyst for the consolidation of standards activity within the mobile data service industry; working in conjunction with other existing standards organizations and industry fora to improve interoperability and decrease operational costs for all involved.
- Provide value and benefits to members in OMA from all parts of the value chain including content and service providers, information technology providers, mobile operators and wireless vendors such that they elect to actively participate in the organization.

*Source: http://openmobilealliance.org/about-oma/*

### OMA LightweightM2M v1.0

The motivation of LightweightM2M is to develop a fast deployable client-server specification to provide machine to machine service.

LightweightM2M is principly a device management protocol, but it should be designed to be able to extend to meet the requirements of applications. LightweightM2M is not restricted to device management, it should be able transfer service / application data.

LightweightM2M implements the interface between M2M device and M2M Server. It provides a choice for the M2M Service Provider to deploy a M2M system to provide service to the M2M user.

Source: http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-lightweightm2m-v1-0

## Leshan

Leshan is an OMA Lightweight M2M (LWM2M) implementation in Java.

Eclipse Leshan relies on the Eclipse IoT Californium project for the CoAP and DTLS implementation.

*Source: https://projects.eclipse.org/projects/iot.leshan*

As you can see, Leshan is built on the base of Californium and CoAP with whom you should be familiar with since the 2nd laboratory of this course. Californium is a project, which its goal is to implement CoAP for Java.

# Scenario of this laboratory

In this laboratory, the following components are provided:

- Leshan server (Java application) in the laboratory network (172.16.32.5)
- Leshan clients (Java application) in the laboratory network (172.16.32.0/24)

The following illustration shows the basic scenario of this laboratory:

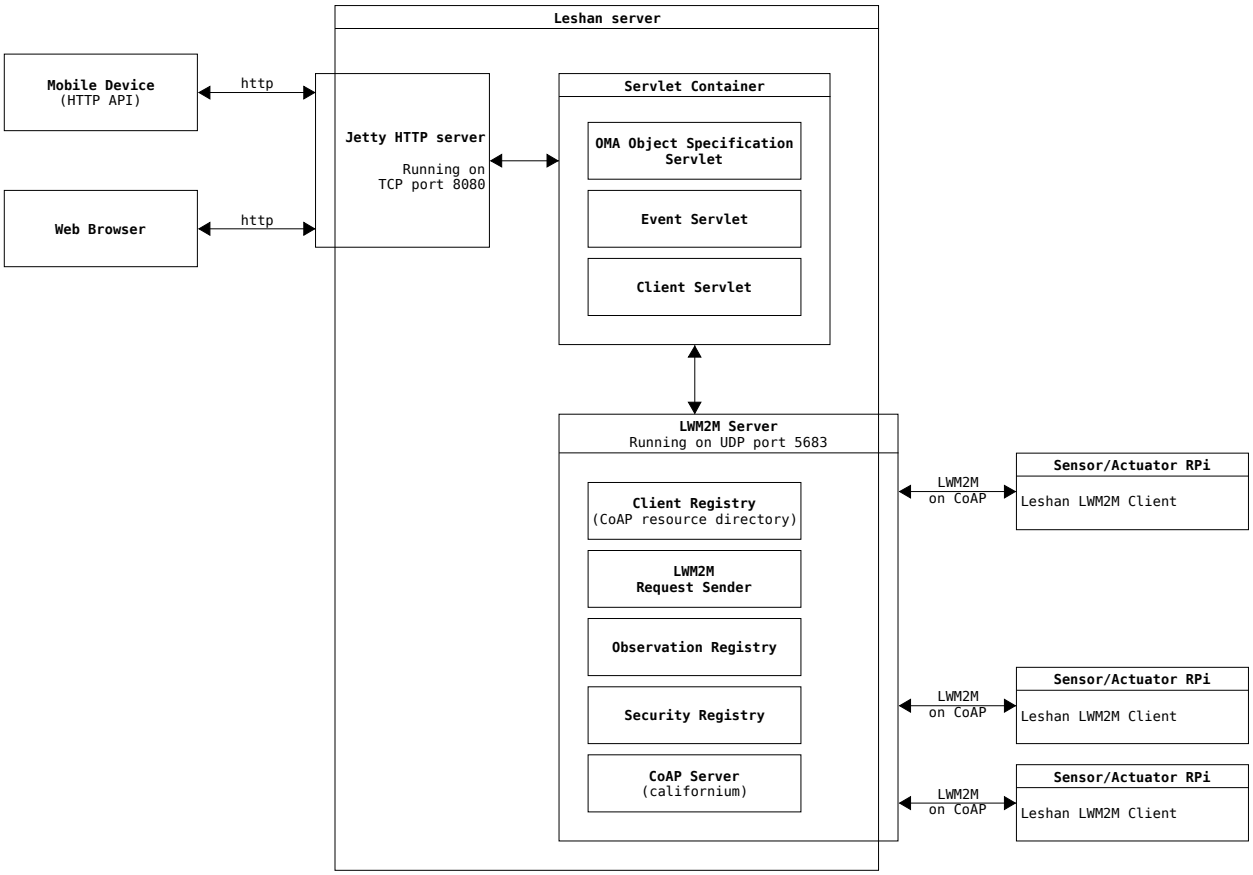


*Illustration 1: Leshan server interoperation*

In this laboratory all the parts are already provided except the server access on the HTTP side. The aim is to implement a simple HTTP client in Python which requests read and/or write operations on the sensor/actuator devices.

# Procedure

The laboratory infrastructure provides a server which has the role of a connection point for all the HTTP requests. This server can be queried, to obtain all the IoT devices that are logged in to the server. Once chosen an IoT device, one can browse the OMA objects that are available on the specific IoT device (sensors and/or actuators).

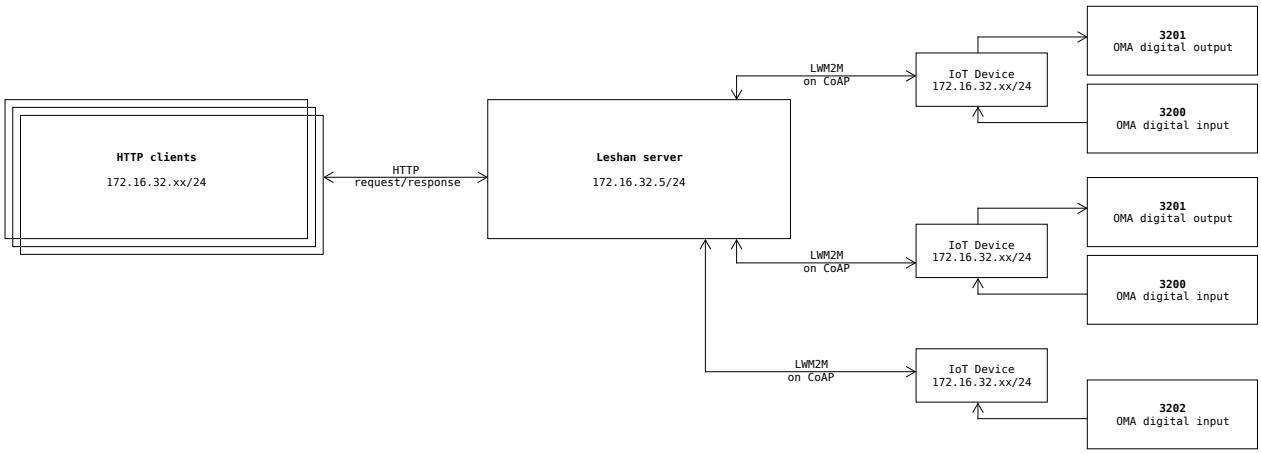


*Illustration 2: Network topology with exemplary IoT devices connected to the Leshan server*

## Available IoT devices in TE523

The IoT devices that are connected to the Leshan server are identified through a name.

The currently available names of the IoT devices are:

**iot-pi-right-back** (IP: 172.16.32.79)

**iot-pi-right-middle** (IP: 172.16.32.73)

**iot-pi-right-front** (IP: 172.16.32.83)

**iot-pi-left-back** (IP: 172.16.32.72)

**iot-pi-left-middle** (IP: 172.16.32.84)

**iot-pi-left-front** (IP: 172.16.32.63)

The devices are also labeled with these names.

## Naming authority

As depicted in Illustration 2 the sensors and actuators are classified as an OMA object ID.

OMA specifies a naming authority under the name OMNA (Open Mobile Naming Authority). The naming is primary built on IDs. You can browse the specifications by visiting the following link (we use the registry "OMA Lightweight M2M (LWM2M)"):

*http://technical.openmobilealliance.org/Technical/technical-information/omna/lightweight-m2m-lwm2m-object-registry*

## Mapping to the Leshan core

These specifications are mapped to the Leshan core and are stored as a JSON representation. Consult the file **"oma-objects-spec.json"** which is also provided with this laboratory. Under the ID **3201**, for example, you will find the following JSON representation:

```
  ...
  {
    "name": "IPSO Digital Output",
    "id": 3201,
    "instancetype": "multiple",
    "mandatory": false,
    "description": "Generic digital output for non-specific actuators",
    "resourcedefs": [
      {
        "id": 5550,
        "name": "Digital Output State",
        "operations": "RW",
        "instancetype": "single",
        "mandatory": true,
        "type": "boolean",
        "range": "",
        "units": "",
        "description": "The current state of a digital output."
      },
      {
        "id": 5551,
        "name": "Digital Output Polarity",
        "operations": "RW",
        "instancetype": "single",
        "mandatory": false,
        "type": "boolean",
        "range": "",
        "units": "",
         "description": "The polarity of a digital output as a Boolean (0 \u003d Normal,
1\u003d Reversed)."
      },
      {
        "id": 5750,
        "name": "Application Type",
        "operations": "RW",
        "instancetype": "single",
        "mandatory": false,
        "type": "string",
        "range": "",
        "units": "",
          "description": "The application type of the output as a string, for instance,
“LED”"
      }
    ]
  },
  ...
```

JSON representation of a generic digital output, stored in the Leshan core.

# Implementation of a simple HTTP client

In this chapter, your task is to write a simple HTTP client in Python to interact with the Leshan server.

## Testing the access to the IoT devices

Firstly, to see if you have connectivity to the Leshan server, open up a web browser of your liking and enter the following address: **`http://172.16.32.5:8080`**

If the call of this page succeeded, you should now see the starting page of the Leshan server. There should appear a list of the IoT devices that are currently connected to the Leshan server, see Available IoT devices in TE523. Click on such a device and try read and/write values to the connected sensors/actuators via the Web-GUI. (Hint: Boolean values have to be submitted as text string: **`"True"/"False"`**)

## Implementing an automated client

The Leshan server provides an API to access the connected IoT which are connected to the server. Furthermore, the API allows it to modify resources (sensors and actuators) that belong to a specific IoT device.

First of all you should think about what your HTTP client should do. E. g. read an analog input value from a sensor at IoT device A and trigger an actuator at IoT device B if the value of the sensor exceeds a defined threshold value.

This laboratory comes with a Python code template where you can implement your client. The template contains already exemplary code snippets for getting a specific sensor value from an IoT device.

The server URL for the API entry point is: **`http://172.16.32.5:8080/api/clients`**

The API serves all its information in the JSON data format. Python serves built in libraries which ease the parsing and accessing of JSON objects significantly. The mapping of JSON objects looks as follows:

| JSON | Python |
|---|---|
| object | dict |
| array | list |
| string | str |
| number (int) | int |
| number (real) | float |
| true | True |
| false | False |
| null | None |

How the JSON strings and Python dictionaries are be transformed can be seen in the functions: **`example_get( url )`** and **`example_put( value, url, resource_id )`** of the code template.

## Code examples

### URL format

An API-URL to access a specific sensor/actuator from the Leshan server looks as follows:

```
http://172.16.32.5:8080/api/clients/iot-pi-right-front/3202/0/5600
http://172.16.32.5:8080/api/clients/iot-pi-right-front/3201/0/5550
```

Explanation:

Server: 172.16.32.5,
Port: 8080,
Client name: iot-pi-right-front,
OMA-Object ID: 3202
Instance number on server: 0,
Attribute-ID: 5600

Hint: To obtain all the resources (and instances) from a specific IoT device you can print the JSON object tree from a specific device:

```
source_tree_client = \
  example_get( http://172.16.32.5:8080/api/clients/iot-pi-right-front )
print( json.dumps( source_tree_client, indent = 4, sort_keys = True ) )
```

### Getting a sensor value

After the call of `example_get( url )` of a specific a sensor we get a dictionary that has the following structure:

```
{
    "content": {
        "id": 5600,
        "value": 443.0
    },
    "status": "CONTENT"
}
```

If one wants to assign the sensor value to a variable, this can be done, writing:

```
data = example_get( url )
sensor_value = data[ 'content' ][ 'value' ]
```

### Writing an actuator value

The opposite of reading a sensor value is writing a value to an actuator.

```
example_put( value, url, attribute_id )
```

Have a look at the `example_put( value, url, attribute_id )` function in the code template. There you can see how the JSON dictionary is set up and dumped to a JSON string. This string is then sent to the Leshan server as a HTTP PUT request.

# Capture LWM2M / CoAP traffic

Now that you have a working client application that communicates with the IoT devices through the Leshan server, you can capture the network traffic on the lines which lead to / from the IoT devices (Raspberry Pi's). To achieve this go to a hub, where an IoT device that is used by your application is connected to. Connect your machine (or a laboratory PC) to the hub and start capturing the traffic with your capturing software of choice (e.g.: Whireshark, tshark, tcpdump, …).

**Hint:** To see all the traffic which is floating through the hub, the capturing interface must be set to **promiscuous mode**. It is the default for Wireshark, but can also be set manually in the capturing options. To be able to operate a network interface in promiscuous mode, in most of the cases, you need administrative privileges on the capturing machine.

## Setup / Scenario

The following figure shows the setup in order to be able to capture traffic from and/or towards the IoT devices (Raspberry Pi's).

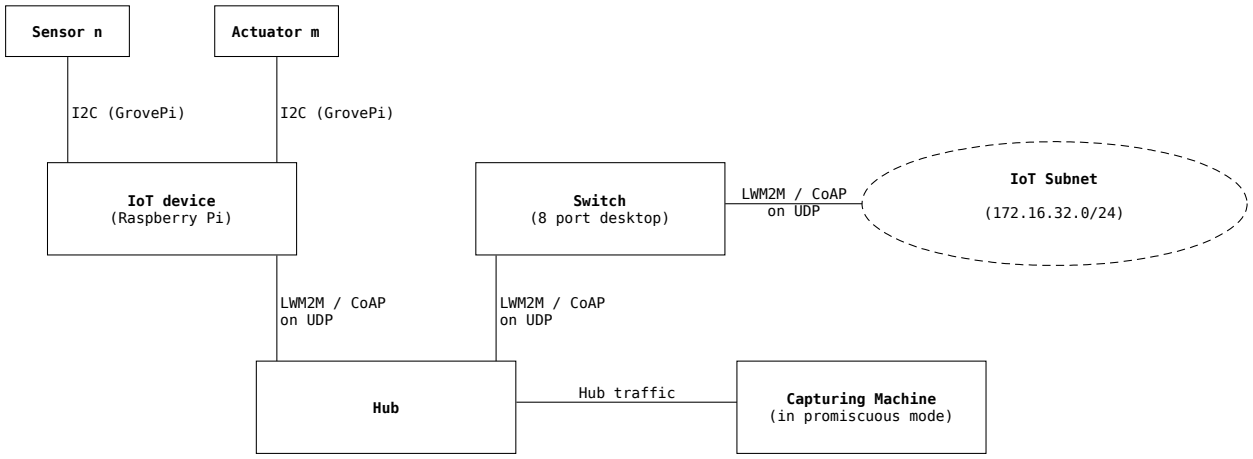


*Illustration 3: Generic scenario to capture LWM2M / CoAP traffic from / to the IoT device*

Your task is now to find / identify and investigate the network traffic (CoAP / LWM2M) which was generated by the Leshan server in order to forward the HTTP control requests that you programmed in your HTTP client application.