IoT Middleware Report

Creation and management of applications using IoT standards and platforms

Zennaro Thomas, 5ISS - B1

November 21, 2020

Contents

Introduction

This report brings together all the practical work carried out under the IoT Middleware module. I first present the application I developed with the Arduino software to highlight the use of the MQTT protocol. Next, I present the creation of the sensors/actuators architecture using the OM2M platform and the associated HTTP requests. Finally, I present the different high-level applications that I was able to create with the Node-Red editor.

1 TP1 : MQTT Protocol

1.1 State of the Art of the MQTT protocol

The architecture of such IoT system is a central broker between publishers and subscribers. The broker is a common interface for sensor devices to connect to and exchange data between them.

MQTT is based on TCP/IP protocol. As it utilizes TCP connection on the transport layer for connections between sensors we deduce that communications are reliable because TCP requires the connection on both sides (emitter and receiver). MQTT is a lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. This protocol offers a quality-of-service for data delivery.

There exist 3 versions of MQTT : 5.0, 3.1.1 and 3.1.

We can find an authentication process : the MQTT broker may require identifiers authentication (username + password) from clients to connect for security. To ensure privacy of messages, the TCP connection may be encrypted with SSL/TLS. Additional security can be added by an application encrypting data, but it is not integrated in MQTT protocol, in order to keep it simple and lightweight.

Use case : *Suppose you have devices that include one button, one light and luminosity sensor. You would like to create a smart system for you house with this behavior:*
• you would like to be able to switch on the light manually with the button
• the light is automatically switched on when the luminosity is under a certain value
What different topics will be necessary to get this behavior and what will the connection be in terms of publishing or subscribing?

The different topics that we can depicted are as follow:
`home/luminosity`
`home/buttonLightState`

Messages come from publishers, through the MQTT broker, to one or more subscribers using these topics.

1.2 Creation of an IoT device with the nodeMCU board that uses MQTT communication

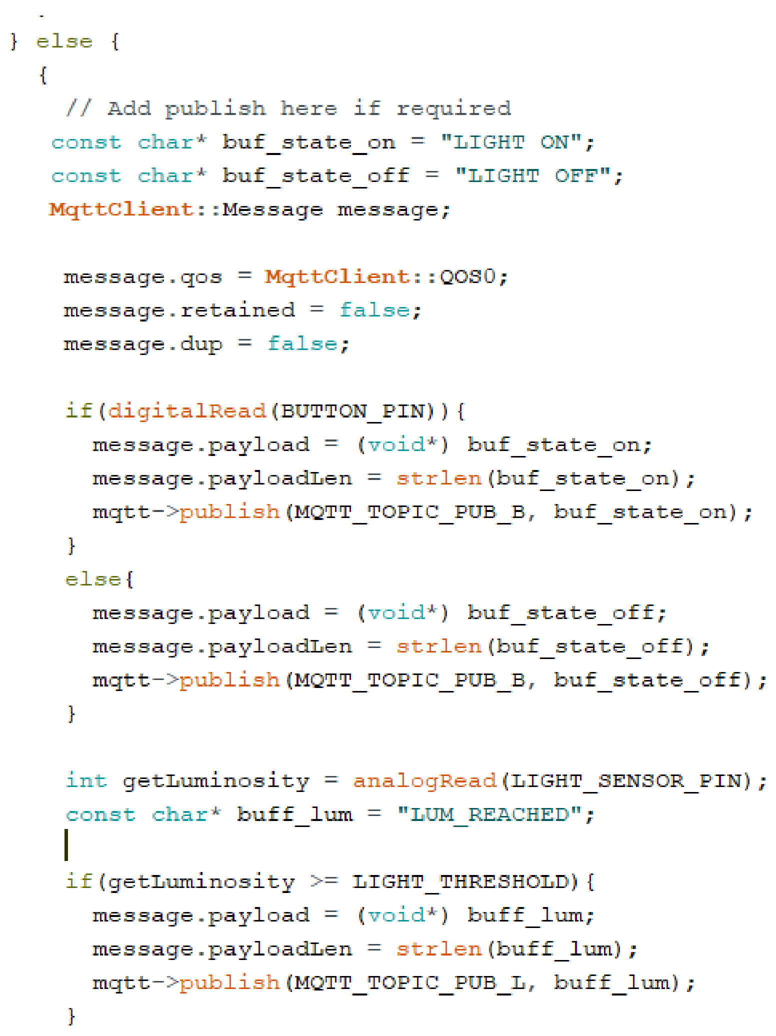
The NodeMCU (Node MicroController Unit) is an open source software and hardware development environment. We program it in low-level machine instructions.
NodeMCU/ESP8266 has 17 GPIO pins which can be assigned to functions such as I2C, I2S, UART, SPI, PWM, IR Remote Control, LED Light and Button programmatically. Each GPIO pin can be configured to internal pull-up or pull-down or set to high impedance.
We can program the ESP8266 with:

- Lua scripts with Node MCU
- C
- C++ with Arduino IDE

`Application:`
I developed the application's light management behavior through MQTT exchanges as defined on the part **1.1**.
The objective was to create the following two subscriptions:

- Turn off or on the LED when the user presses the button
- send a message via the broker and changed the state of the LED when the light sensor is below a certain predefined threshold

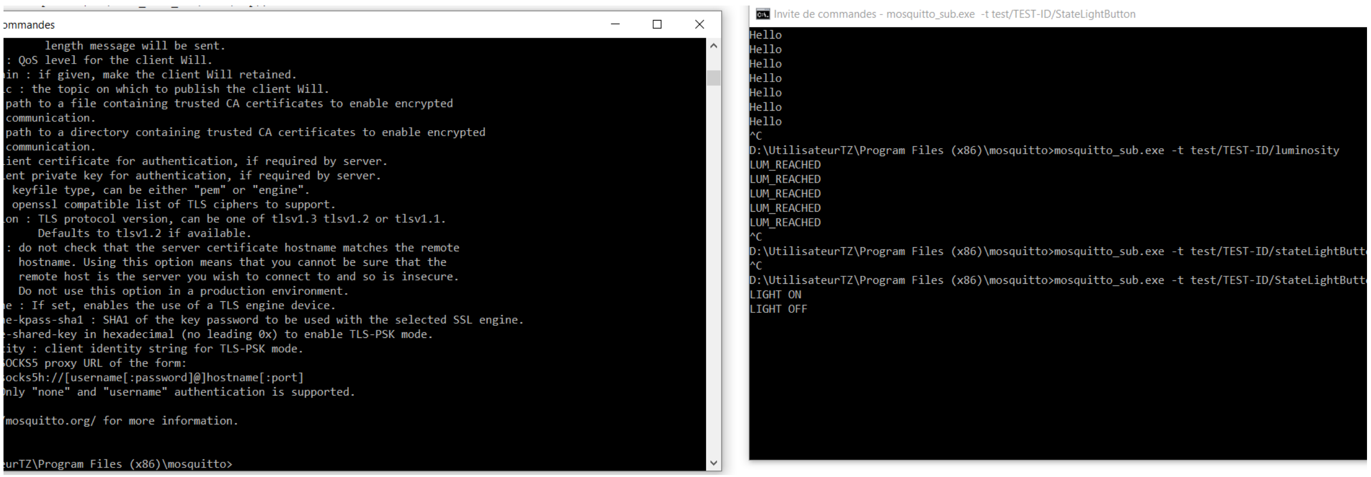
The code is given below.



```
} else {
  {
    // Add publish here if required
   const char* buf_state_on = "LIGHT ON";
   const char* buf_state_off = "LIGHT OFF";
   MqttClient::Message message;

    message.qos = MqttClient::QOS0;
    message.retained = false;
    message.dup = false;

    if(digitalRead(BUTTON_PIN)){
      message.payload = (void*) buf_state_on;
      message.payloadLen = strlen(buf_state_on);
      mqtt->publish(MQTT_TOPIC_PUB_B, buf_state_on);
    }
    else{
      message.payload = (void*) buf_state_off;
      message.payloadLen = strlen(buf_state_off);
      mqtt->publish(MQTT_TOPIC_PUB_B, buf_state_off);
    }

    int getLuminosity = analogRead(LIGHT_SENSOR_PIN);
    const char* buff_lum = "LUM_REACHED";

    if(getLuminosity >= LIGHT_THRESHOLD){
      message.payload = (void*) buff_lum;
      message.payloadLen = strlen(buff_lum);
      mqtt->publish(MQTT_TOPIC_PUB_L, buff_lum);
    }
```

Figure 1: Source Code for the publishers (C++)

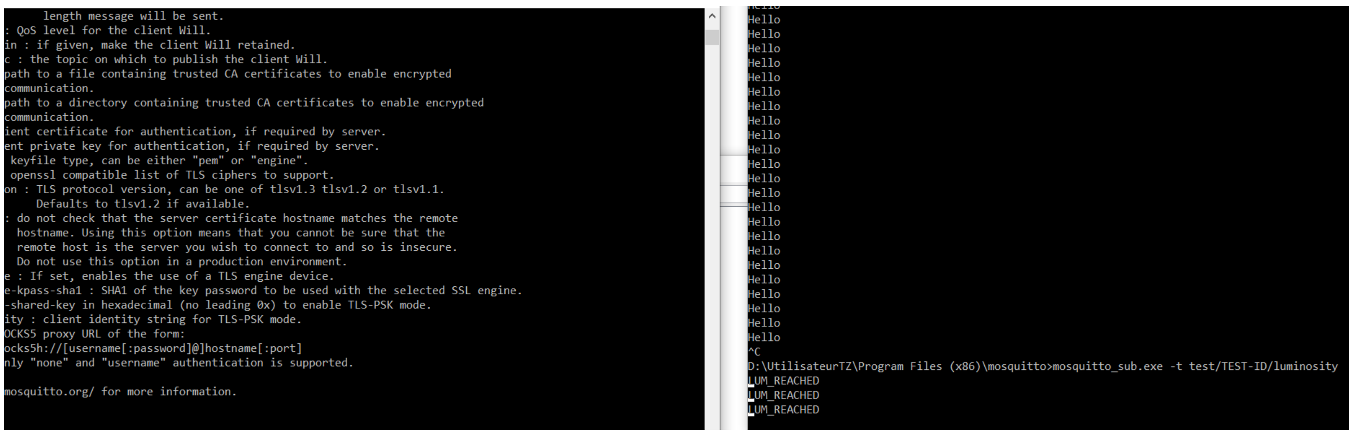
After establishing the connection between mosquitto_pub and the Arduino, I can launch my mosquitto_sub application to subscribe to the 2 topics. Mosquitto will act as a mediator, so messages will be sent periodically (by default every 30 seconds). Every time I press the push button to change the state of the LED, this is kept. A message is sent indicating the current state of the LED (see Figure 2). A similar result is observed for the second topic (see Figure 3) indicating whether or not the room brightness threshold has been reached. For more information, see the full source code in the Appendix.



```
ommandes
       length message will be sent.
: QoS level for the client Will.
in : if given, make the client Will retained.
c : the topic on which to publish the client Will.
path to a file containing trusted CA certificates to enable encrypted
communication.
path to a directory containing trusted CA certificates to enable encrypted
communication.
ient certificate for authentication, if required by server.
ent private key for authentication, if required by server.
 keyfile type, can be either "pem" or "engine".
 openssl compatible list of TLS ciphers to support.
on : TLS protocol version, can be one of tlsv1.3 tlsv1.2 or tlsv1.1.
     Defaults to tlsv1.2 if available.
: do not check that the server certificate hostname matches the remote
  hostname. Using this option means that you cannot be sure that the
  remote host is the server you wish to connect to and so is insecure.
  Do not use this option in a production environment.
e : If set, enables the use of a TLS engine device.
e-kpass-sha1 : SHA1 of the key password to be used with the selected SSL engine.
-shared-key in hexadecimal (no leading 0x) to enable TLS-PSK mode.
ity : client identity string for TLS-PSK mode.
OCKS5 proxy URL of the form:
ocks5h://[username[:password]@]hostname[:port]
nly "none" and "username" authentication is supported.

mosquitto.org/ for more information.


urTZ\Program Files (x86)\mosquitto>
```

```
Invite de commandes - mosquitto_sub.exe  -t test/TEST-ID/StateLightButton
Hello
Hello
Hello
Hello
Hello
Hello
Hello
^C
D:\UtilisateurTZ\Program Files (x86)\mosquitto>mosquitto_sub.exe -t test/TEST-ID/luminosity
LUM_REACHED
LUM_REACHED
LUM_REACHED
LUM_REACHED
LUM_REACHED
^C
D:\UtilisateurTZ\Program Files (x86)\mosquitto>mosquitto_sub.exe -t test/TEST-ID/stateLightButt
^C
D:\UtilisateurTZ\Program Files (x86)\mosquitto>mosquitto_sub.exe -t test/TEST-ID/StateLightButt
LIGHT ON
LIGHT OFF
```

Figure 2: Exchanged communication during the first subscription



```
       length message will be sent.
: QoS level for the client Will.
in : if given, make the client Will retained.
c : the topic on which to publish the client Will.
path to a file containing trusted CA certificates to enable encrypted
communication.
path to a directory containing trusted CA certificates to enable encrypted
communication.
ient certificate for authentication, if required by server.
ent private key for authentication, if required by server.
 keyfile type, can be either "pem" or "engine".
 openssl compatible list of TLS ciphers to support.
on : TLS protocol version, can be one of tlsv1.3 tlsv1.2 or tlsv1.1.
     Defaults to tlsv1.2 if available.
: do not check that the server certificate hostname matches the remote
  hostname. Using this option means that you cannot be sure that the
  remote host is the server you wish to connect to and so is insecure.
  Do not use this option in a production environment.
e : If set, enables the use of a TLS engine device.
e-kpass-sha1 : SHA1 of the key password to be used with the selected SSL engine.
-shared-key in hexadecimal (no leading 0x) to enable TLS-PSK mode.
ity : client identity string for TLS-PSK mode.
OCKS5 proxy URL of the form:
ocks5h://[username[:password]@]hostname[:port]
nly "none" and "username" authentication is supported.

mosquitto.org/ for more information.
```

```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
^C
D:\UtilisateurTZ\Program Files (x86)\mosquitto>mosquitto_sub.exe -t test/TEST-ID/luminosity
LUM_REACHED
LUM_REACHED
LUM_REACHED
```

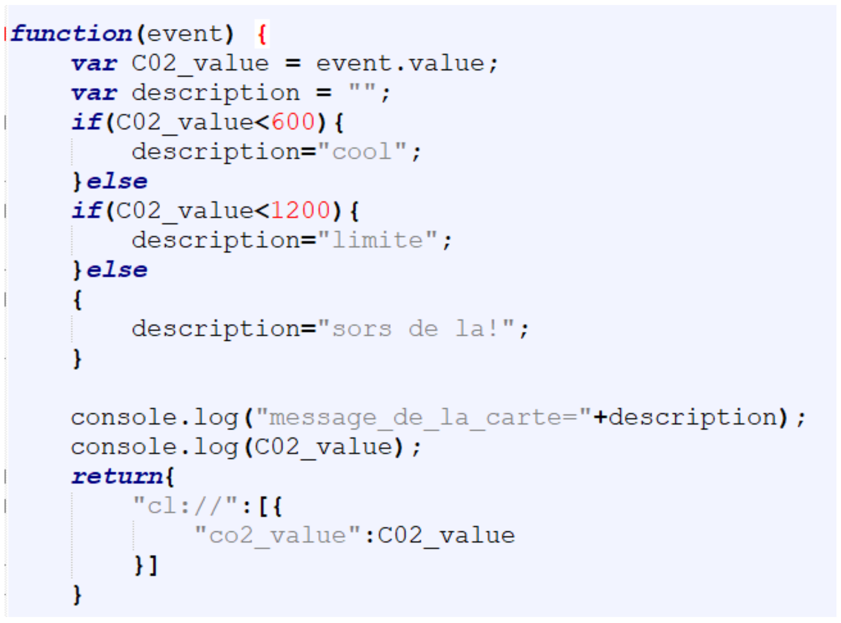
Figure 3: Exchanged communication during the second subscription

2 TP2 : Handling of Octave and a mangOH yellow IoT solution development board

During the second lab we studied mangOH yellow board. mangOH is a family of open source hardware platforms for the IoT. This development board includes many modules and components such as different sensors (air, brightness...), an ARM processor, a modem to wirelessly connect its IoT application on a mobile network. It also allows us to access services associated with the Cloud but also communication services such as sending and receiving SMS.

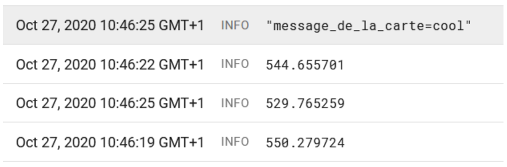
The map resources can be managed via the web interface provided by Octave. With this interface, you activate the resources you want and then create an observation on the sensor in question. An observation will be able to propose resources, streams, edge actions, or local data storage. Moreover, we specify the period for which we will connect the data and we can filter this data (by applying buffering, filtering, throttling operations). I created an observation on the light sensor for which I only recovered the values above a certain threshold.

The final objective was to create my own application which was intended to collect the CO2 rate in the room air and then return a message estimating the air quality (i.e., poor, normal, optimal). I realized an edge action on the resource associated with the CO2 sensor (see Figure 4). The transmitted information is then retrieved from the interface console (see Figure 5). Looking at the default stream, you can see the values that are sent to the cloud (Figure 6).



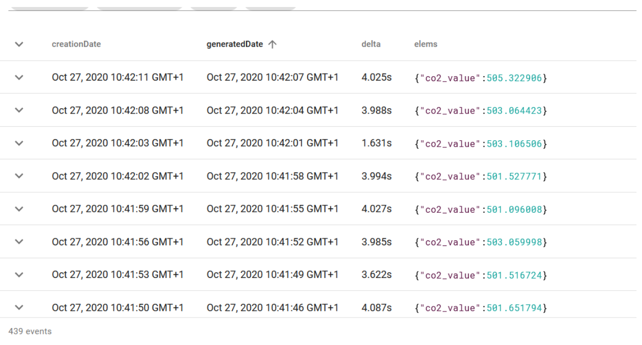
```
function(event) {
    var C02_value = event.value;
    var description = "";
    if(C02_value<600){
        description="cool";
    }else
    if(C02_value<1200){
        description="limite";
    }else
    {
        description="sors de la!";
    }

    console.log("message_de_la_carte="+description);
    console.log(C02_value);
    return{
        "cl://":[{
            "co2_value":C02_value
        }]
    }
```

Figure 4: Message received for the first subscription



```
Oct 27, 2020 10:46:25 GMT+1   INFO   "message_de_la_carte=cool"
Oct 27, 2020 10:46:22 GMT+1   INFO   544.655701
Oct 27, 2020 10:46:25 GMT+1   INFO   529.765259
Oct 27, 2020 10:46:19 GMT+1   INFO   550.279724
```

Figure 5: Information of the air quality retrieved on the console



creationDate	generatedDate ↑	delta	elems
Oct 27, 2020 10:42:11 GMT+1	Oct 27, 2020 10:42:07 GMT+1	4.025s	{"co2_value":505.322906}
Oct 27, 2020 10:42:08 GMT+1	Oct 27, 2020 10:42:04 GMT+1	3.988s	{"co2_value":503.064423}
Oct 27, 2020 10:42:03 GMT+1	Oct 27, 2020 10:42:01 GMT+1	1.631s	{"co2_value":503.106506}
Oct 27, 2020 10:42:02 GMT+1	Oct 27, 2020 10:41:58 GMT+1	3.994s	{"co2_value":501.527771}
Oct 27, 2020 10:41:59 GMT+1	Oct 27, 2020 10:41:55 GMT+1	4.027s	{"co2_value":501.096008}
Oct 27, 2020 10:41:56 GMT+1	Oct 27, 2020 10:41:52 GMT+1	3.985s	{"co2_value":503.059998}
Oct 27, 2020 10:41:53 GMT+1	Oct 27, 2020 10:41:49 GMT+1	3.622s	{"co2_value":501.516724}
Oct 27, 2020 10:41:50 GMT+1	Oct 27, 2020 10:41:46 GMT+1	4.087s	{"co2_value":501.651794}

439 events

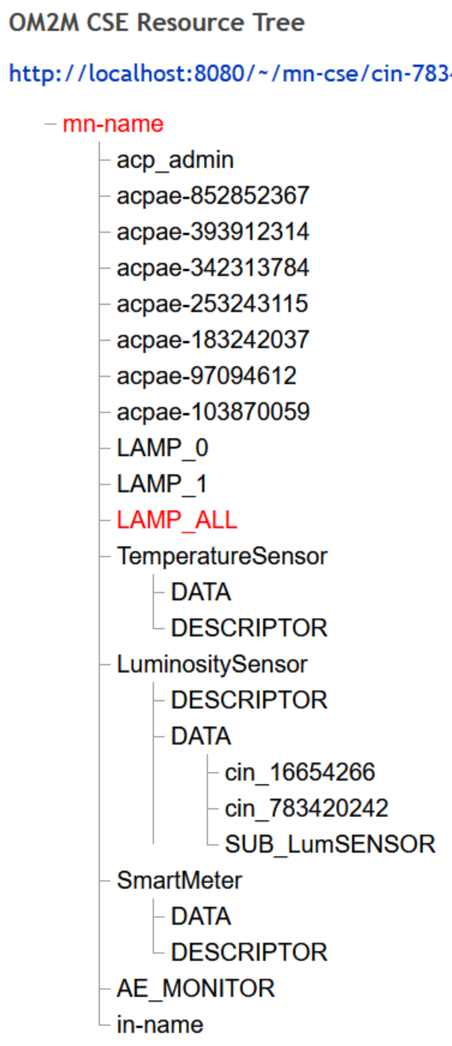
Figure 6: Values of CO2 sent to the Cloud

3 TP3 : Deployment of an architecture using the oneM2M standard

3.1 Deployment of the Architecture

I created 3 AE matching to 3 sensors. Indeed, we had a Smart Meter, a luminosity sensor and a temperature sensor. For each of them, I added 2 containers :

- a DESCRIPTOR container which which brings together all the characteristics of the sensor
- a DATA container which contains the different retrieved values

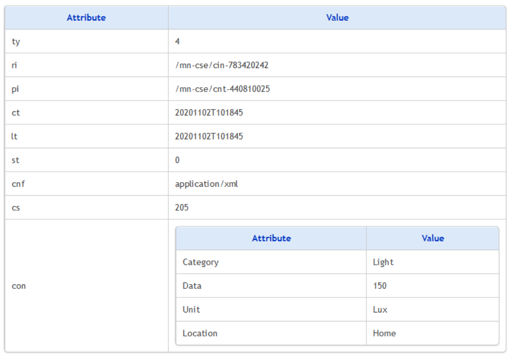


```
OM2M CSE Resource Tree

http://localhost:8080/~/mn-cse/cin-783420242

- mn-name
   ├ acp_admin
   ├ acpae-852852367
   ├ acpae-393912314
   ├ acpae-342313784
   ├ acpae-253243115
   ├ acpae-183242037
   ├ acpae-97094612
   ├ acpae-103870059
   ├ LAMP_0
   ├ LAMP_1
   ├ LAMP_ALL
   ├ TemperatureSensor
   │   ├ DATA
   │   └ DESCRIPTOR
   ├ LuminositySensor
   │   ├ DESCRIPTOR
   │   ├ DATA
   │   │   ├ cin_16654266
   │   │   ├ cin_783420242
   │   │   └ SUB_LumSENSOR
   ├ SmartMeter
   │   ├ DATA
   │   └ DESCRIPTOR
   ├ AE_MONITOR
   └ in-name
```

Figure 7: Resources tree on the OM2M platform

Then, I created a new AE with a request reachability attribute (rr) at true and a point of access (poa) with the url of a monitor, so that this AE represents the monitoring application. You can visualize the whole architecture on Figure 7.
By allowing this AE to subscribe to AEs referring to previous sensors, it is possible to listen to the arrival of new sensor values on a terminal.

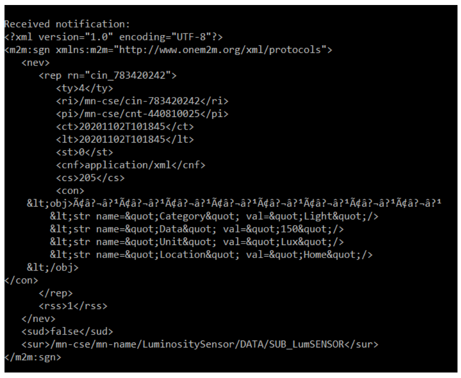
Then I created a new content instance for the DATA container of the light AE, in order to test the operation of subscription. It works well as you can see on both Figure 8 and Figure 9.



Attribute	Value
ty	4
ri	/mn-cse/cin-783420242
pi	/mn-cse/cnt-440810025
ct	20201102T101845
lt	20201102T101845
st	0
cnf	application/xml
cs	205
con	

Attribute	Value
Category	Light
Data	150
Unit	Lux
Location	Home

Figure 8: New CIN created on the OM2M platform

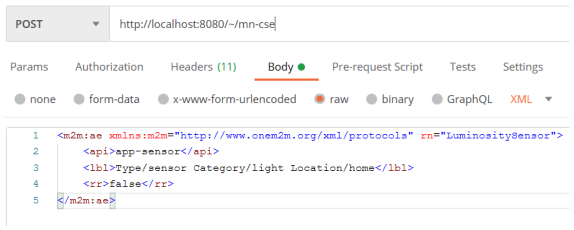


```
Received notification:
<?xml version="1.0" encoding="UTF-8"?>
<m2m:sgn xmlns:m2m="http://www.onem2m.org/xml/protocols">
   <nev>
      <rep rn="cin_783420242">
         <ty>4</ty>
         <ri>/mn-cse/cin-783420242</ri>
         <pi>/mn-cse/cnt-440810025</pi>
         <ct>20201102T101845</ct>
         <lt>20201102T101845</lt>
         <st>0</st>
         <cnf>application/xml</cnf>
         <cs>205</cs>
         <con>
    &lt;obj>Ã¢â?¬â?¹Ã¢â?¬â?¹Ã¢â?¬â?¹Ã¢â?¬â?¹Ã¢â?¬â?¹Ã¢â?¬â?¹Ã¢â?¬â?¹Ã¢â?¬â?¹
        &lt;str name=&quot;Category&quot; val=&quot;Light&quot;/>
        &lt;str name=&quot;Data&quot; val=&quot;150&quot;/>
        &lt;str name=&quot;Unit&quot; val=&quot;Lux&quot;/>
        &lt;str name=&quot;Location&quot; val=&quot;Home&quot;/>
    &lt;/obj>
</con>
      </rep>
      <rss>1</rss>
   </nev>
   <sud>false</sud>
   <sur>/mn-cse/mn-name/LuminositySensor/DATA/SUB_LumSENSOR</sur>
</m2m:sgn>
```

Figure 9: Retrieve of the new light CIN by listening on the port 1400

3.2 Main requests used

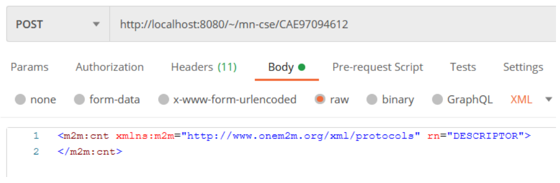
Here I show you the typical queries I have developed with Postman client to feed the resource tree and perform different operations.



POST http://localhost:8080/~/mn-cse

```
<m2m:ae xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="LuminositySensor">
    <api>app-sensor</api>
    <lbl>Type/sensor Category/light Location/home</lbl>
    <rr>false</rr>
</m2m:ae>
```

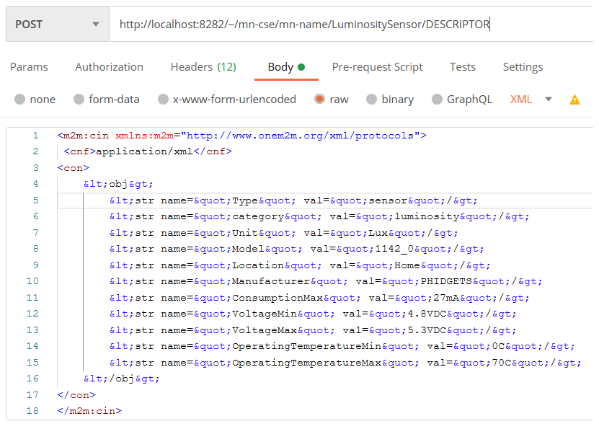
Figure 10: Request for the creation of a new AE named LuminositySensor



POST http://localhost:8080/~/mn-cse/CAE97094612

```
<m2m:cnt xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="DESCRIPTOR">
</m2m:cnt>
```

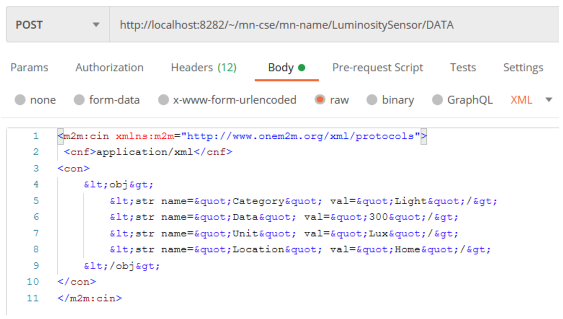
Figure 11: Request for the creation of a CNT named DESCRIPTOR



POST http://localhost:8282/~/mn-cse/mn-name/LuminositySensor/DESCRIPTOR

```
<m2m:cin xmlns:m2m="http://www.onem2m.org/xml/protocols">
 <cnf>application/xml</cnf>
<con>
    &lt;obj&gt;
        &lt;str name=&quot;Type&quot; val=&quot;sensor&quot;/&gt;
        &lt;str name=&quot;category&quot; val=&quot;luminosity&quot;/&gt;
        &lt;str name=&quot;Unit&quot; val=&quot;Lux&quot;/&gt;
        &lt;str name=&quot;Model&quot; val=&quot;1142_0&quot;/&gt;
        &lt;str name=&quot;Location&quot; val=&quot;Home&quot;/&gt;
        &lt;str name=&quot;Manufacturer&quot; val=&quot;PHIDGETS&quot;/&gt;
        &lt;str name=&quot;ConsumptionMax&quot; val=&quot;27mA&quot;/&gt;
        &lt;str name=&quot;VoltageMin&quot; val=&quot;4.8VDC&quot;/&gt;
        &lt;str name=&quot;VoltageMax&quot; val=&quot;5.3VDC&quot;/&gt;
        &lt;str name=&quot;OperatingTemperatureMin&quot; val=&quot;0C&quot;/&gt;
        &lt;str name=&quot;OperatingTemperatureMax&quot; val=&quot;70C&quot;/&gt;
    &lt;/obj&gt;
</con>
</m2m:cin>
```

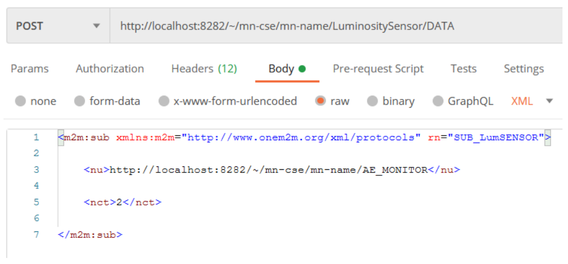
Figure 12: Request for the creation of a new CIN for the DESCRIPTOR container



POST http://localhost:8282/~/mn-cse/mn-name/LuminositySensor/DATA

```
<m2m:cin xmlns:m2m="http://www.onem2m.org/xml/protocols">
 <cnf>application/xml</cnf>
<con>
    &lt;obj&gt;
        &lt;str name=&quot;Category&quot; val=&quot;Light&quot;/&gt;
        &lt;str name=&quot;Data&quot; val=&quot;300&quot;/&gt;
        &lt;str name=&quot;Unit&quot; val=&quot;Lux&quot;/&gt;
        &lt;str name=&quot;Location&quot; val=&quot;Home&quot;/&gt;
    &lt;/obj&gt;
</con>
</m2m:cin>
```

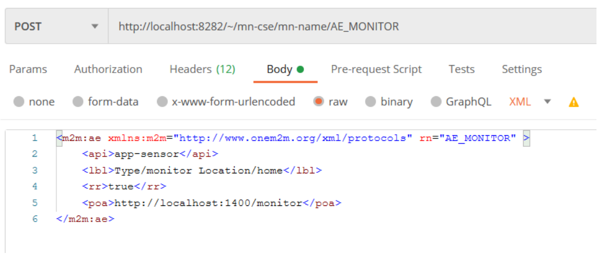
Figure 13: Request for the creation of a new CIN for the DATA container



POST http://localhost:8282/~/mn-cse/mn-name/LuminositySensor/DATA

```xml
<m2m:sub xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="SUB_LumSENSOR">

    <nu>http://localhost:8282/~/mn-cse/mn-name/AE_MONITOR</nu>

    <nct>2</nct>

</m2m:sub>
```

Figure 14: Request for the creation of subscription of the Light DATA container



POST http://localhost:8282/~/mn-cse/mn-name/AE_MONITOR

```xml
<m2m:ae xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="AE_MONITOR" >
    <api>app-sensor</api>
    <lbl>Type/monitor Location/home</lbl>
    <rr>true</rr>
    <poa>http://localhost:1400/monitor</poa>
</m2m:ae>
```

Figure 15: Request for the creation of a new AE representing the Monitoring application

3.3 Comparison between MQTT and oneM2M

One difference between MQTT and oneM2M is in the resource tree structure. Indeed, for MQTT, the resource tree must be properly designed in order to be able to adapt to any evolution of new topics that may appear in the life of the project. In addition, when using MQTT, operation and URI information must be added to the publication content. On the contrary for HTTP (used by oneM2M), we will define operations (GET, POST, PUT, DELETE...) that allow to characterize the nature of the request. It is combined with a URI that determines which resource you want to send your request to. In addition, to perform a publish/subscribe operation, simply enter the type in the request header.

Criteria	**MQTT**	**oneM2M**
standard coverage	storage management of data exchanged between the connected things	Services as discovery, communication and data, network, items management
deployment model	One server for MQTT messages and a Publish/Subscription concept for clients	Based on 3 layers (application layer, service layer, network layer) ; Deployed model for things/gateways/the Cloud
data model	Based on the use of topics (hierarchically)	Structured data thanks to a resources tree and a REST architecture
hardware platform and programming language	No specific material for the client/server management ; languages such as Lua, Java	Several open-source platforms ; languages such as Java, Python...
security	Possible communication encryption (but heavy)	Authentication, communication encryption

Table 1: Comparison between MQTT and oneM2M using some criteria

3.4 Positioning between Octave Sierra Wireless and OM2M

Octave Sierra Wireless is a Cloud solution very simple of use with a large variety of options. The interface is very intuitive.

Creating new virtual resources does not require a HTTP REST request as is the case with the OM2M platform. In addition, Octave is optimal for device management, evolution of services and resources, ideal for the maintainability of the Cloud architecture.

With Octave, we can see the evolution of data from resources graphically but also given by data. We can easily filter the ones we want to store on the Cloud.

With OM2M platform, we can easily have a overview of the resources tree and the hierarchy dependencies. We need to use a client (like Postman) to manage the resources with operation like GET, POST, PUT, DELETE...With Octave, everything is done through the web interface.

4 TP4 : Fast application prototyping for IoT

The source code of each flow is given in the following file: *flows_TP4_Zennaro.json*.

4.1 Applications realized with Node-red interface

4.1.1 Application n°1

The objective was to retrieve get sensor values or actuator state (lamps...). I choose to get data instances of *LAMP_0* and of the *TemperatureSensor* that I created on TP3.

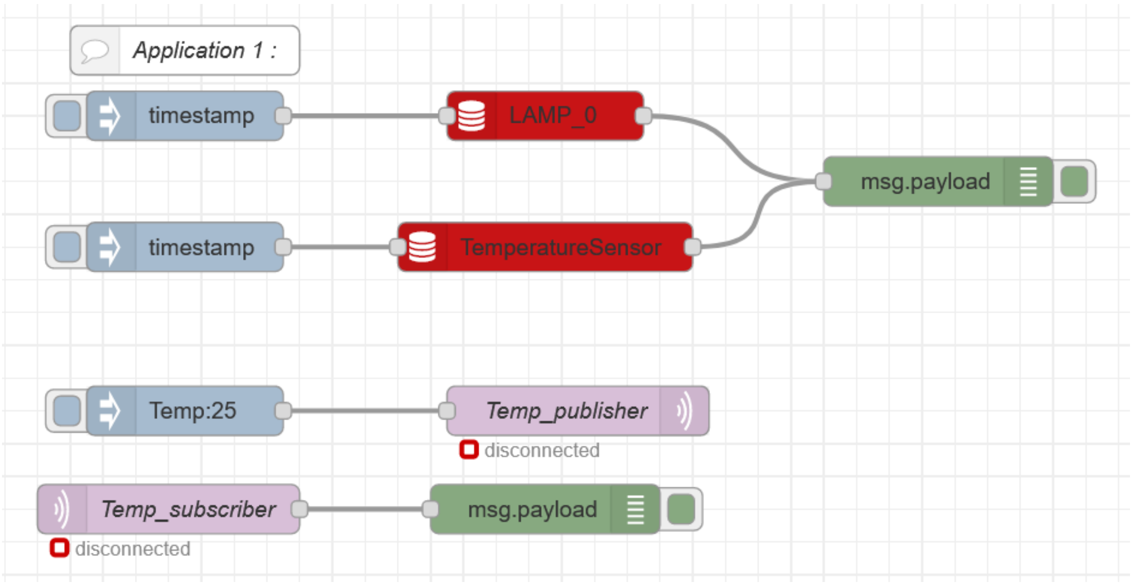


Figure 16: Flow for the first application

4.1.2 Application n°2

The objective was to perform a simple test between luminosity value and a threshold using a function node for instance.Depending on the previous result I can either turn off or turn on both lamps.

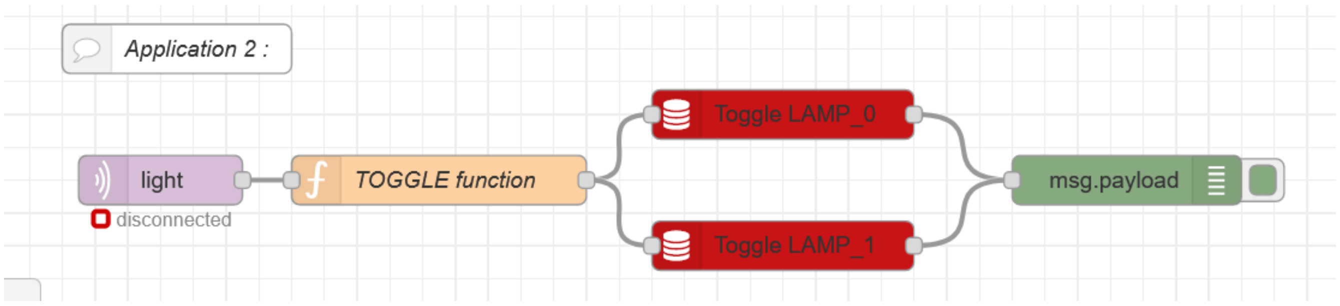


Figure 17: Flow for the second application

4.1.3 Application n°3

The objective was to create a dashboard that plots values of the *TemperatureSensor* on a chart (see Figure 18). Moreover, with a switch node I was able to change the state of *LAMP_0* (see Figure 19). The result dashboard is given on Figure 20. You can access to it by typing on the URL bar: *http://localhost:1880/ui*.

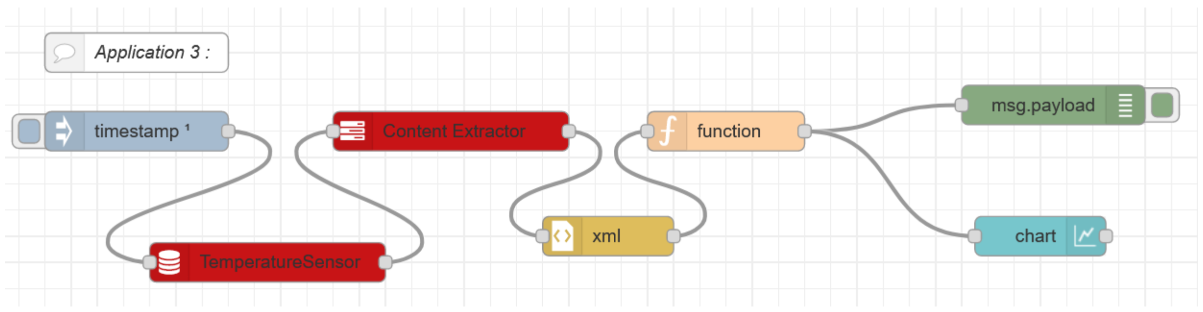


Figure 18: Flow for the third application regarding the chart

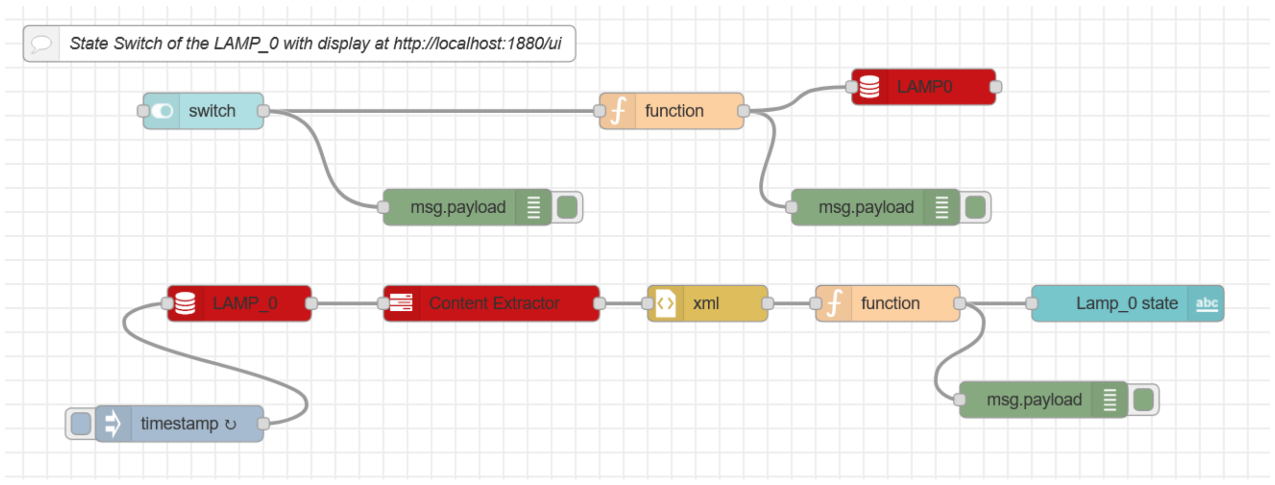


Figure 19: Flow for the third application regarding the switch

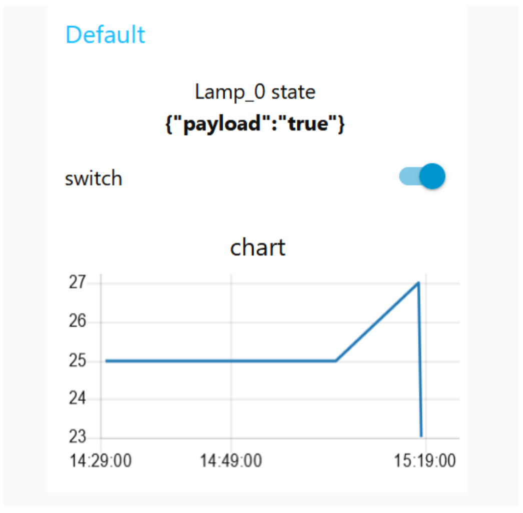


Figure 20: Dashboard

4.1.4 Application n°4

For the last application, I decided to write the state of *LAMP_0* directly on a file each time I inject a data.

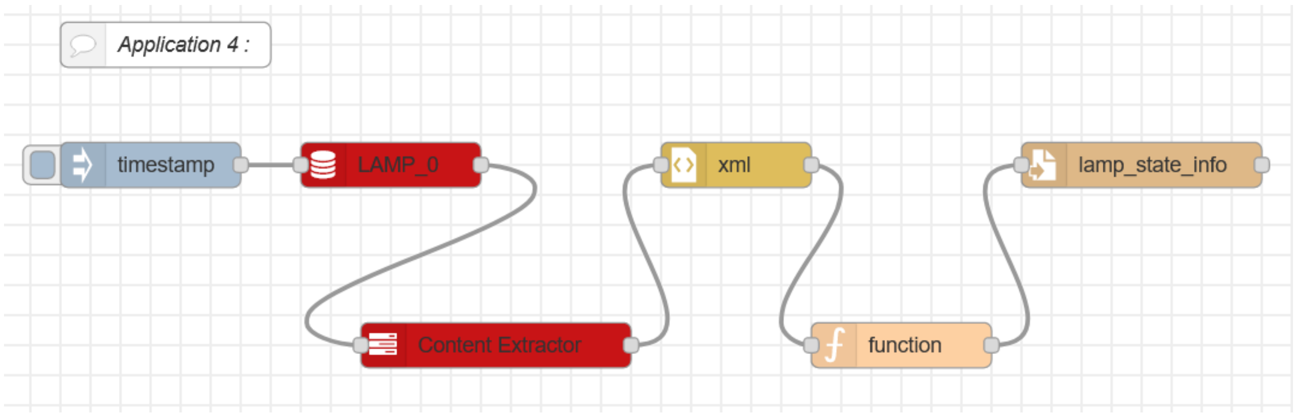


Figure 21: Flow for the fourth application

4.2 Benefits of Node-red

Node-red makes it possible to build applications intuitively and visually with its graphical interface. There are many libraries and nodes available to us (MQTT, dashboard...). Node-red provides us with a wide range of connectors, processing components that can be connected directly to wires via a "drag and drop" system. We can process the data directly, implement functions in order to vary the parameters of captors or actuators (change the state of a lamp...)

Moreover, the deployment of our flows is quite practical and we can:

- deploy all flows
- specifically choose a flow that you want to deploy
- deploy only flows that have undergone changes

A debug console is also integrated which allows testing directly on the interface. Therefore, IoT applications can be easily implemented and the OM2M platform can be used after installing the corresponding nodes in Node-red.

4.3 Drawbacks of Node-red

Node-red will however present some disadvantages. Indeed, one recovers directly in output the payload of each message but it can for example become complicated to filter the data of such or such sensor. Moreover, in terms of security it is not optimal. Indeed, anyone who can retrieve its IP address can access the editor and so he would be able to do changes regarding our flows. This is only suitable if we run Node-red on a trusted network.

Conclusion

To conclude, this set of labs allowed me to put into practice the knowledge developed through the MOOC on the oneM2M standard and the OM2M platform. Moreover, the first lab allowed the publication/subscription mechanism to be manipulated through MQTT and the mosquitto broker. Then, with Node-Red, we can develop applications faster with an intuitive way. So all these tools are practical in order to realize high-level applications to interact with sensors and actuators and can therefore be the basis of the creation of IoT solutions.

List of Figures

5 Arduino application source code

```
#include <Arduino.h>
#include <ESP8266WiFi.h>

// Enable MqttClient logs
#define MQTT_LOG_ENABLED 1
// Include library
#include <MqttClient.h>


#define LOG_PRINTFLN(fmt, ...)  logfln(fmt, ##__VA_ARGS__)
#define LOG_SIZE_MAX 128
void logfln(const char *fmt, ...) {
  char buf[LOG_SIZE_MAX];
  va_list ap;
  va_start(ap, fmt);
  vsnprintf(buf, LOG_SIZE_MAX, fmt, ap);
  va_end(ap);
  Serial.println(buf);
}

#define HW_UART_SPEED                  115200L
#define MQTT_ID                       "TEST-ID"
#define SENSOR_LIGHT_PIN
#define BUTTON_PIN
#define LED_PIN
#define LUM_THRESHOLD 300

static MqttClient *mqtt = NULL;
static WiFiClient network;
const char* MQTT_TOPIC_SUB = "test"/MQTT_ID/"sub";
const char* MQTT_TOPIC_PUB_L = "luminosity";
const char* MQTT_TOPIC_PUB_B = "StateLightButton";

// ============== Object to supply system functions ============================
class System: public MqttClient::System {
public:

  unsigned long millis() const {
    return ::millis();
  }

  void yield(void) {
    ::yield();
  }
};

// ============== Setup all objects ============================================
void setup() {
  // Setup hardware serial for logging
  Serial.begin(HW_UART_SPEED);
  while (!Serial);

  // Setup WiFi network
  WiFi.mode(WIFI_STA);
  WiFi.hostname("ESP_" MQTT_ID);
  WiFi.begin("Cisco38658");
  LOG_PRINTFLN("\n");
  LOG_PRINTFLN("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    LOG_PRINTFLN(".");
  }
  LOG_PRINTFLN("Connected to WiFi");
  LOG_PRINTFLN("IP: %s", WiFi.localIP().toString().c_str());

  // Setup MqttClient
  MqttClient::System *mqttSystem = new System;
  MqttClient::Logger *mqttLogger = new MqttClient::LoggerImpl<HardwareSerial>(Serial)
      ;
  MqttClient::Network * mqttNetwork = new MqttClient::NetworkClientImpl<WiFiClient>(
      network, *mqttSystem);
  //// Make 128 bytes send buffer
  MqttClient::Buffer *mqttSendBuffer = new MqttClient::ArrayBuffer<128>();
  //// Make 128 bytes receive buffer
  MqttClient::Buffer *mqttRecvBuffer = new MqttClient::ArrayBuffer<128>();
  //// Allow up to 2 subscriptions simultaneously
  MqttClient::MessageHandlers *mqttMessageHandlers = new MqttClient::
      MessageHandlersImpl<2>();
  //// Configure client options
  MqttClient::Options mqttOptions;
  ////// Set command timeout to 10 seconds
  mqttOptions.commandTimeoutMs = 10000;
  //// Make client object
  mqtt = new MqttClient(
    mqttOptions, *mqttLogger, *mqttSystem, *mqttNetwork, *mqttSendBuffer,
    *mqttRecvBuffer, *mqttMessageHandlers
  );

  pinMode(LIGHT_SENSOR_PIN, INPUT);
  pinMode(BUTTON_PIN, INPUT);
  pinMode(LED_PIN, OUTPUT);


}

// ============== Subscription callback ========================================
void processMessage(MqttClient::MessageData& md) {
  const MqttClient::Message& msg = md.message;
  char payload[msg.payloadLen + 1];
  memcpy(payload, msg.payload, msg.payloadLen);
  payload[msg.payloadLen] = '\0';
  LOG_PRINTFLN(
    "Message arrived: qos %d, retained %d, dup %d, packetid %d, payload:[%s]",
    msg.qos, msg.retained, msg.dup, msg.id, payload
  );
}
// ============== Main loop ====================================================
void loop() {
  // Check connection status
  if (!mqtt->isConnected()) {
    // Close connection if exists
    network.stop();
    // Re-establish TCP connection with MQTT broker
    LOG_PRINTFLN("Connecting");
    network.connect("192.168.1.131", 1883);
    if (!network.connected()) {
      LOG_PRINTFLN("Can't establish the TCP connection");
      delay(5000);
      ESP.reset();
    }
    // Start new MQTT connection
    MqttClient::ConnectResult connectResult;
    // Connect
    {
      MQTTPacket_connectData options = MQTTPacket_connectData_initializer;
      options.MQTTVersion = 4;
      options.clientID.cstring = (char*)MQTT_ID;
      options.cleansession = true;
      options.keepAliveInterval = 15; // 15 seconds
      MqttClient::Error::type rc = mqtt->connect(options, connectResult);
      if (rc != MqttClient::Error::SUCCESS) {
        LOG_PRINTFLN("Connection error: %i", rc);
        return;
      }
    }
    {
      // Add subscribe here if required
      MqttClient::Error::type rc = mqtt->subscribe(
       MQTT_TOPIC_SUB, MqttClient::QOS0, processMessage
      );
      if (rc != MqttClient::Error::SUCCESS) {
        LOG_PRINTFLN("Subscribe error: %i", rc);
        LOG_PRINTFLN("Drop connection");
        mqtt->disconnect();
        return;
      }
    }
  } else {
    {
      // Add publish here if required
     const char* buf_state_on = "LIGHT ON";
     const char* buf_state_off = "LIGHT OFF";
     MqttClient::Message message;

      message.qos = MqttClient::QOS0;
      message.retained = false;
      message.dup = false;

      if(digitalRead(BUTTON_PIN)){
        message.payload = (void*) buf_state_on;
        message.payloadLen = strlen(buf_state_on);
        mqtt->publish(MQTT_TOPIC_PUB_B, buf_state_on);
      }
      else{
        message.payload = (void*) buf_state_off;
        message.payloadLen = strlen(buf_state_off);
        mqtt->publish(MQTT_TOPIC_PUB_B, buf_state_off);
      }

      int getLuminosity = analogRead(LIGHT_SENSOR_PIN);
      const char* buff_lum = "LUM_REACHED";

      if(getLuminosity >= LIGHT_THRESHOLD){
        message.payload = (void*) buff_lum;
        message.payloadLen = strlen(buff_lum);
        mqtt->publish(MQTT_TOPIC_PUB_L, buff_lum);
      }

    }
    // Idle for 30 seconds
    mqtt->yield(30000L);
  }
}
```