Practical Sheet Nº 4 (MQTT)

Objectives	Duration
At the end of the training, you will be able to: • Understand what an M2M communications protocol is for • Understand the operation mode of a messaging protocol considering the publish/subscribe model. • Know the operation mode of the MQTT standard. • Implement an M2M communications system using the MQTT standard.	60min.

Introduction

The MQTT (Message Queue Telemetry Transport) protocol was created by IBM in the late 1990s. Its initial application was to integrate sensors in pipelines with satellite communications. It is a messaging protocol that supports asynchronous communications between parties. The MQTT protocol uses a publish and subscribe model having become an open OASIS standard in late 2014 and is supported by multiple programming languages and several open-source implementations exist.

MQTT is a machine-to-machine (M2M) communications protocol with wide application in the domain of Cyber-Physical Systems or the Internet of Things. It is designed to allow message transport in publish/subscribe mode, as can be seen in Figure 1.

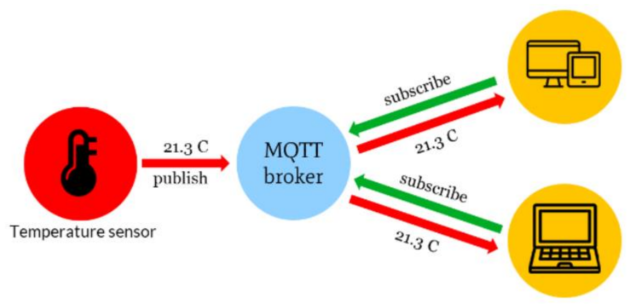


Figure 1: Data Flow in the MQTT protocol, from sensor (machine) to client (machine)

MQTT is a valuable protocol for communication in low-bandwidth and resource-constrained scenarios. Its small protocol stack, low power consumption, small data packets, and efficient message distribution make it ideal for limited devices. MQTT is lightweight and flexible, making it suitable for high hardware constraints, high latency, and limited bandwidth networks. It emerged to overcome the limitations of the widely used HTTP protocol in web services, namely:

- HTTP is a synchronous protocol, i.e., the client always waits for the server to respond.
- HTTP is a one-way protocol, i.e., the client must always initiate the connection.
- HTTP is a 1-1 protocol, i.e., the client makes a request and the server replies.
- HTTP is a heavy protocol with many headers and rules and is not suitable for narrow networks.

For the above reasons, most scalable high-performance systems use an asynchronous messaging system instead of web services for data exchange.

On the other hand, the MQTT protocol requires much less network and device resources to operate with it publish/subscribe model, which allows you to fully disaggregate the data provider and the data consumer.

The publish/subscribe model.

The MQTT protocol defines two types of entities in the network: a message broker and clients. The broker is a server that receives messages from clients and forwards them only to clients that have subscribed to them.

A client is any device that can interact with the broker to send and receive messages. A client can be an IoT device or an application in a data centre that processes data, as can be seen in Figure 2.

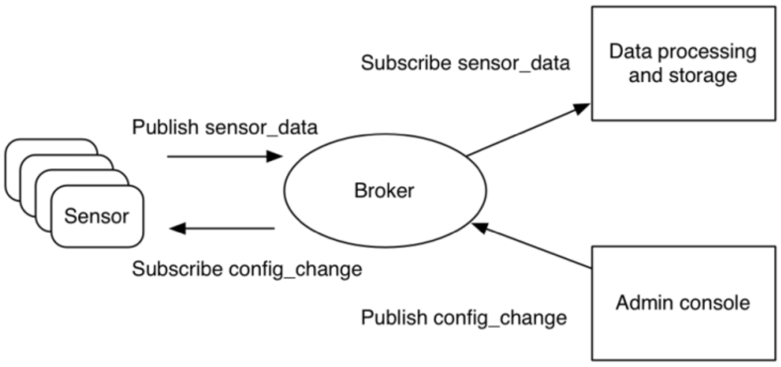


Figure 2: MQTT Publish/Subscribe model for IoT applications.

The main steps in the MQTT communication process are as follows:

1. The client connects to the broker and can subscribe to any specific message thread. This connection can be of the TCP/IP type or simply an encrypted TLS connection.
2. The client publishes messages related to a specific topic by sending the message and the topic to the broker.
3. The broker forwards the message to all clients that subscribed to that topic.

Since MQTT messages are organized by topics, application development is more flexible since it is enough to specify that certain clients can only interact with certain messages. In the example in Figure2, the sensors publish the readings in the "sensor_data" topic and subscribe to the "config_change" topic. The "Data Processing and Storage" application that stores the data coming from the sensors in a database subscribe to the "sensor_data" topic. The "Admin Console" application can receive commands from the system administrator to adjust sensor settings, such as sensitivity and sampling frequency, and publish these changes to the "config_change" topic.

The MQTT protocol has a fixed header with only 2 bytes (red zone) where the where the message type is specified.

Then the variable header zone is specified, where the topic is identified in text (green zone) relative to the message.

Finally, the binary zone related to the payload (blue zone) is defined. Any data format can be used in encoding the payload (JSON, XML, encrypted binary or Base64), if the target clients are able to parse this payload.

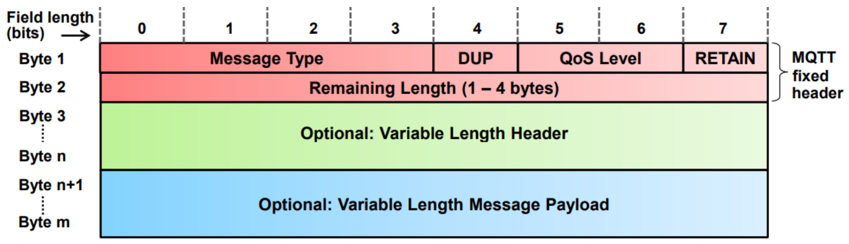


Figure 3: Generic MQTT message format

Structure of a payload message

The message payload in MQTT can contain an ASCII-encoded Ad-Hoc format. For instance, it can follow the message example bellow:

msg

ATTRIBUTE1= VALUE1 & ATTRIBUTE2 = VALUE2& ...& ATTRIBUTE X= VALUEX

In the frame entered before, the following characters are used to identify:

'&' – ATTRIBUTE/VALUE pair separator

'=' – Divider of the ATTRIBUTE/VALUE pair

Main operation modes of the MQTT protocol: Push vs. Pull

In this practical sheet we intend to use an MQTT server (broker) that will be locally accessible on the lab's LAN. The ESP32 DEV KIT will be used to implement the clients. Figure 4 shows a diagram with several clients exchanging messages through an MQTT broker. As in most publish/subscribe systems, sending messages involves posting on one or several specified topics. Whenever it receives a new message, the broker forwards this to all subscribers of that topic. In this work we intend to use two modes of interaction, Push and Pull, where the broker will process and transmit 3 distinct topics: Topic1, Topic2 and Topic3, as represented in Figure 4.

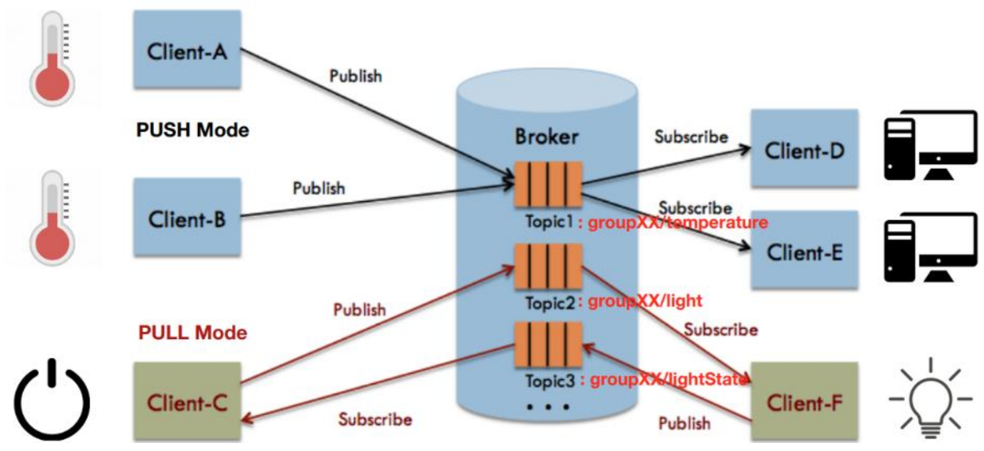


Figure 4: Generic mode of the MQTT message

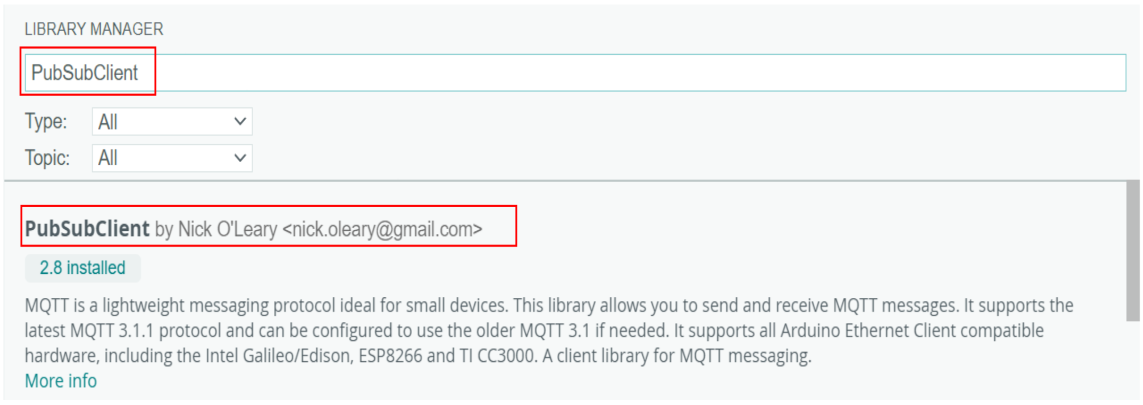
In **Push** mode, Client A and Client B (temperature sensors) publish their messages in Topic 1 (groupXX/temperature). When a message is received at the broker, it is broadcast to all current subscribers of that topic. In this case, Client D and Client E will receive all messages transmitted by Clients A and B . This mode is unidirectional, and the data is "pushed" to the server (push) by the device generating the data. In an IoT context, the device that publishes is usually a sensor node and the subscriber is usually the server that processes and stores the data coming from the sensor.~

In **Pull** mode, Client C sends publishes messages on Topic 2 (groupXX/light) that are forwarded via broker to Client F, which responds by sending a message on Topic 3 (groupXX/lightState). In an IoT context, this mode of operation can be useful, for example, in situations where a client needs to update the state of another client and get a confirmation of that state update, as is the case in the example depicted in Figure 4.

Funded by the European Union

Part 1 - Implementing MQTT communications in Push Mode

1. In the Arduino IDE go to ***Sketch > Include Libraries > Manage Libraries*** and install the MQTT library ***PubSubClient***



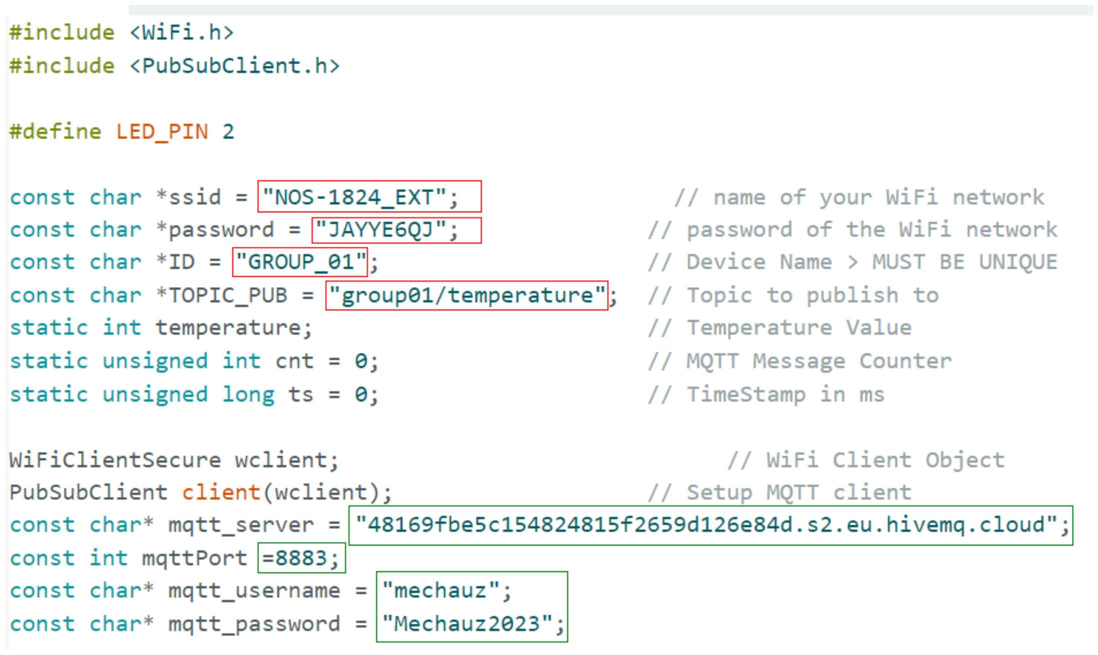
PubSubClient by Nick O'Leary <nick.oleary@gmail.com>

2.8 installed

MQTT is a lightweight messaging protocol ideal for small devices. This library allows you to send and receive MQTT messages. It supports the latest MQTT 3.1.1 protocol and can be configured to use the older MQTT 3.1 if needed. It supports all Arduino Ethernet Client compatible hardware, including the Intel Galileo/Edison, ESP8266 and TI CC3000. A client library for MQTT messaging.

More info

2. Consider the initial configuration code file ***"mqtt_push.ino"*** to implement the MQTT client on ESP32s DEVKIT.
   Replace ***ssid*** and **password** with the correct wi-fi credentials. The ID and the TOPIC_PUB must be associated with the GROUP/KIT number. For Group01, use ***GROUP_01*** for the ***ID*** and ***group01/temperature*** for the ***TOPIC_PUB*** .
   This example is using a cloud broker **HIVEMQ**, but if you want to use a self-hosted, just replace the server's name and, port and credentials.
   Using the text editor of the Arduino IDE create a sketch named ***FP4_EX2_MQTT_client_push.ino.***



```
#include <WiFi.h>
#include <PubSubClient.h>

#define LED_PIN 2

const char *ssid = "NOS-1824_EXT";              // name of your WiFi network
const char *password = "JAYYE6QJ";             // password of the WiFi network
const char *ID = "GROUP_01";                   // Device Name > MUST BE UNIQUE
const char *TOPIC_PUB = "group01/temperature"; // Topic to publish to
static int temperature;                        // Temperature Value
static unsigned int cnt = 0;                   // MQTT Message Counter
static unsigned long ts = 0;                   // TimeStamp in ms

WiFiClientSecure wclient;                          // WiFi Client Object
PubSubClient client(wclient);                  // Setup MQTT client
const char* mqtt_server = "48169fbe5c154824815f2659d126e84d.s2.eu.hivemq.cloud";
const int mqttPort =8883;
const char* mqtt_username = "mechauz";
const char* mqtt_password = "Mechauz2023";
```

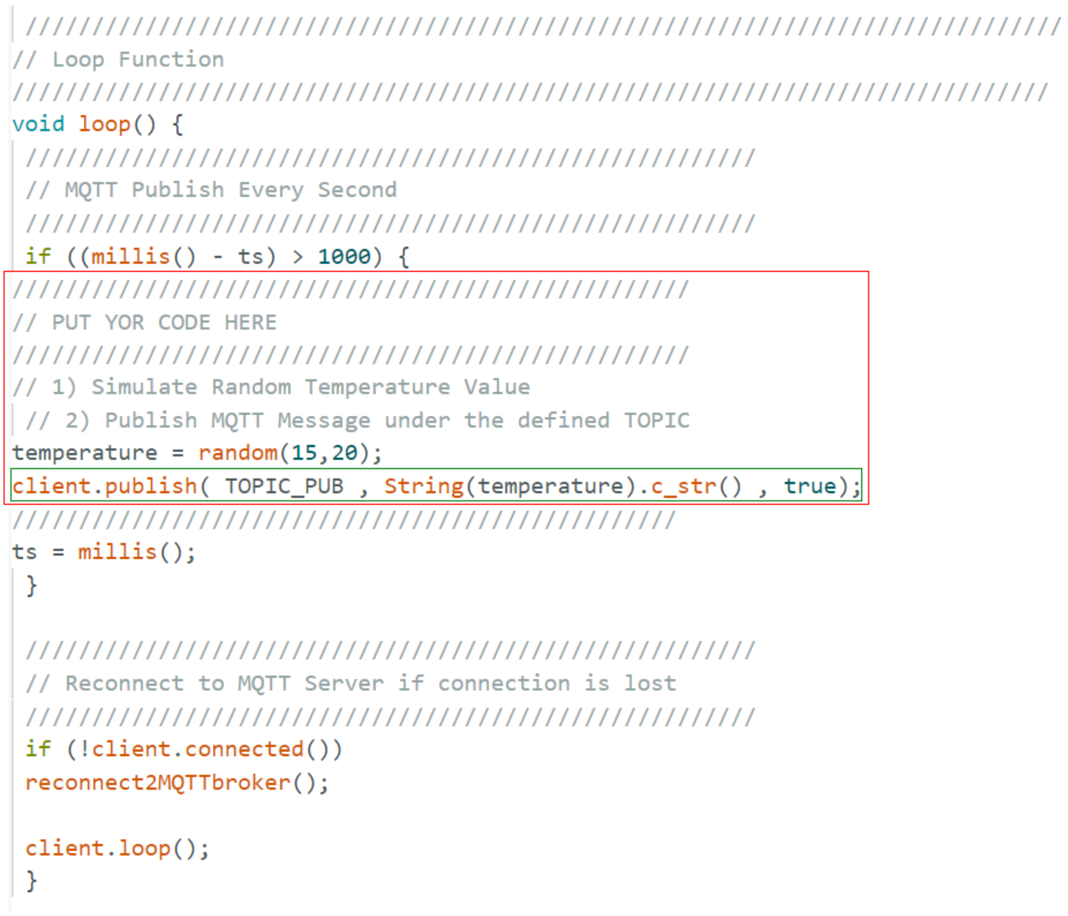
3. The following ***reconnect2MQTTBroker()*** is responsible to maintain the connection to the MQTT Broker.

```
/////////////////////////////////////////////////
// Reconnect to MQTT broker
////////////////////////////////////////////////
void reconnect2MQTTbroker() {
  while (!client.connected()) { // Loop until reconnected
  Serial.print(“Connecting to MQTT broker…”);
  if(client.connect(ID)) { // Attempt to connect
    Serial.println(“Connected to MQTT Broker“);
    Serial.println('\n');
  }
  else {
    Serial.println("Try again in 5 seconds”);
    delay(5000); // Wait 5 seconds before retrying
  }
 }
}
```

4. The function ***setup()*** initiates the WiFi connection and sets the MQTT broker.

```
//////////////////////////////////////////////////////////////////////////////
// Setup Function - Inits Serial Port and WiFi Connection
//////////////////////////////////////////////////////////////////////////////
void setup() {
  Serial.begin(115200); // Init Serial Port
 /////////////////////////////////////////////////////
 // Connect to WiFi
 /////////////////////////////////////////////////////
 Serial.print("Connecting to ");
 Serial.println(ssid);
 WiFi.begin(ssid, password); // Connects to WiFi Network
 while (WiFi.status() != WL_CONNECTED){ // Waits for WiFi connection
 delay(500);
 Serial.print(".");
}
 Serial.println("ESP connected to WiFi with IP address: ");
 Serial.println(WiFi.localIP());
 wclient.setCACert(root_ca);
 /////////////////////////////////////////////////////
 // Set MQTT Broker and Callback Function
 /////////////////////////////////////////////////////
 client.setServer(mqtt_server,mqttPort); // Sets IP+PORT for the MQTT broker (PORT=1883)
 ts = millis(); // ts = TimeStamp
 }
```

5. The ***loop()*** function will be continuously executing code for the MQTT client via the ***client.loop()*** instruction. In addition, a regular check is made on the status of the connection to the MQTT broker. If the connection fails, a new attempt is made to connect to the broker. Consider the following code to implement the loop function:

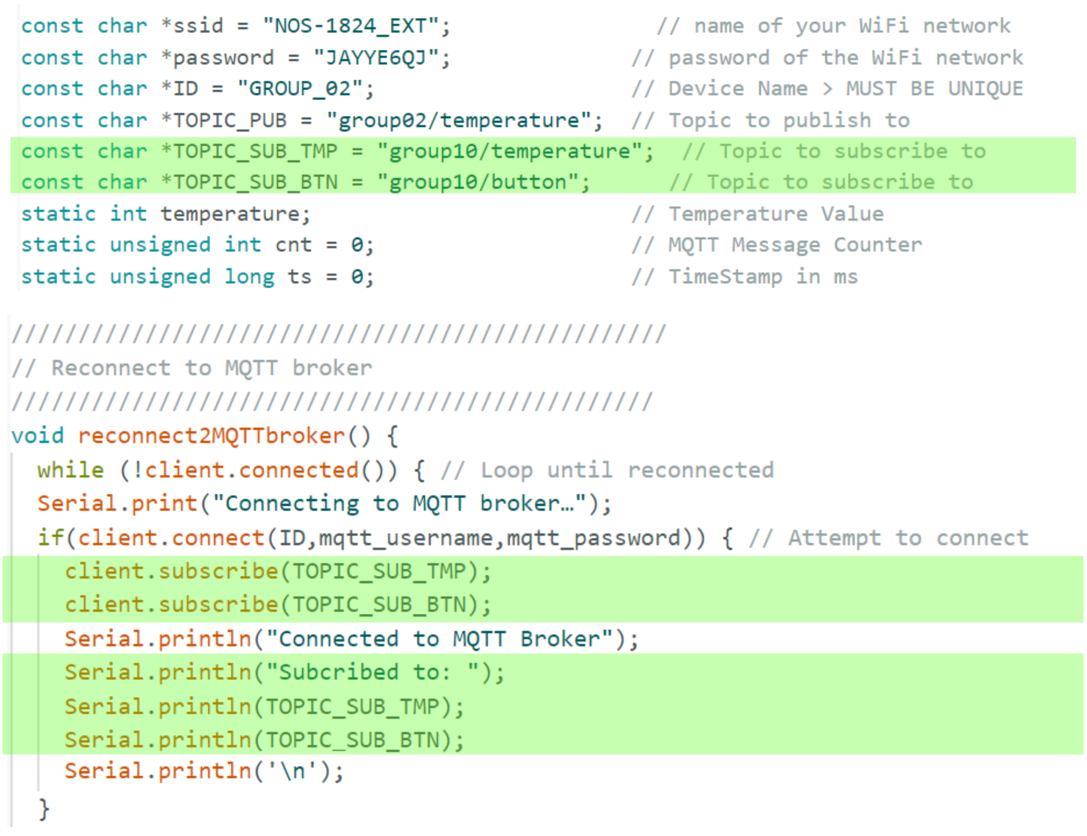


```
 ///////////////////////////////////////////////////////////////////////////////
// Loop Function
///////////////////////////////////////////////////////////////////////////////
void loop() {
 ///////////////////////////////////////////////////////
 // MQTT Publish Every Second
 ///////////////////////////////////////////////////////
 if ((millis() - ts) > 1000) {
///////////////////////////////////////////////////
// PUT YOR CODE HERE
///////////////////////////////////////////////////
// 1) Simulate Random Temperature Value
 // 2) Publish MQTT Message under the defined TOPIC
temperature = random(15,20);
client.publish( TOPIC_PUB , String(temperature).c_str() , true);
//////////////////////////////////////////////////
ts = millis();
 }

 ///////////////////////////////////////////////////////
 // Reconnect to MQTT Server if connection is lost
 ///////////////////////////////////////////////////////
 if (!client.connected())
 reconnect2MQTTbroker();

 client.loop();
 }
```

6. Test the MQTT client code you implemented by connecting to the broker available for that purpose. Ask the trainer for the broker's IP. Look at the dashboard for information regarding your device.

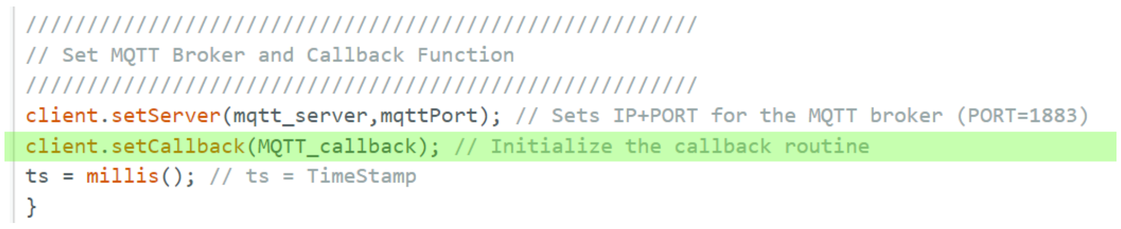
7. Edit the code to generate a random temperature between 15 and 30 degrees and publish it on the same topic every two seconds.

8. Make a copy of the sketch tested earlier and rename it to ***FP4_EX8_MQTT_client_push.ino.*** Using the text editor of the Arduino IDE, adapt the code to subscribe to the following new topics ***"group10/temperature"*** and ***"group10/button".*** To do so, add and comment the code represented by the green lines.



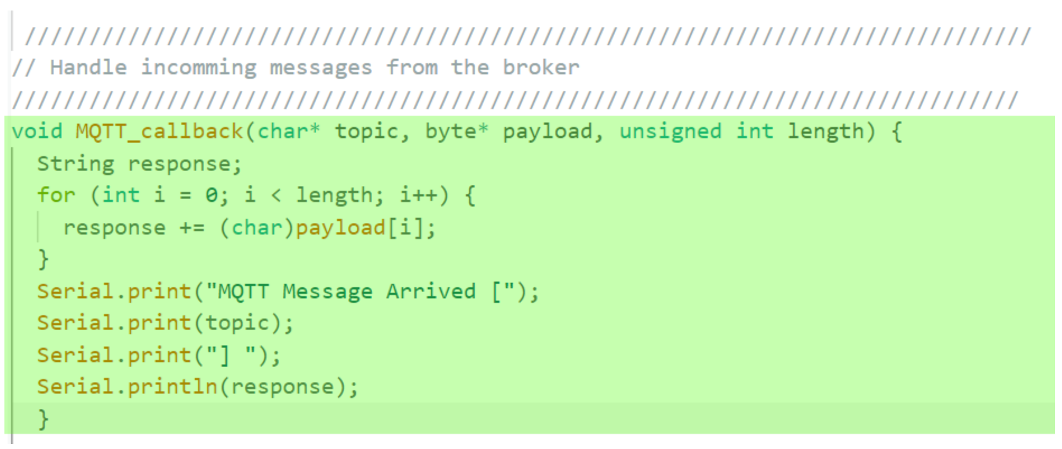
```
const char *ssid = "NOS-1824_EXT";                 // name of your WiFi network
const char *password = "JAYYE6QJ";               // password of the WiFi network
const char *ID = "GROUP_02";                     // Device Name > MUST BE UNIQUE
const char *TOPIC_PUB = "group02/temperature";   // Topic to publish to
const char *TOPIC_SUB_TMP = "group10/temperature";  // Topic to subscribe to
const char *TOPIC_SUB_BTN = "group10/button";      // Topic to subscribe to
static int temperature;                          // Temperature Value
static unsigned int cnt = 0;                     // MQTT Message Counter
static unsigned long ts = 0;                     // TimeStamp in ms

///////////////////////////////////////////////////
// Reconnect to MQTT broker
///////////////////////////////////////////////////
void reconnect2MQTTbroker() {
  while (!client.connected()) { // Loop until reconnected
  Serial.print("Connecting to MQTT broker…");
  if(client.connect(ID,mqtt_username,mqtt_password)) { // Attempt to connect
    client.subscribe(TOPIC_SUB_TMP);
    client.subscribe(TOPIC_SUB_BTN);
    Serial.println("Connected to MQTT Broker");
    Serial.println("Subcribed to: ");
    Serial.println(TOPIC_SUB_TMP);
    Serial.println(TOPIC_SUB_BTN);
    Serial.println('\n');
  }
```

9. Now initialize the callback that handles the incoming messages by adding the following code represented by the green line.



```
/////////////////////////////////////////////////////
// Set MQTT Broker and Callback Function
/////////////////////////////////////////////////////
client.setServer(mqtt_server,mqttPort); // Sets IP+PORT for the MQTT broker (PORT=1883)
client.setCallback(MQTT_callback); // Initialize the callback routine
ts = millis(); // ts = TimeStamp
}
```

10. To process/print the received message to the console add the following callback function, just after the variable definition, which will be triggered whenever the MQTT client receives a new message from the subscribed topics.



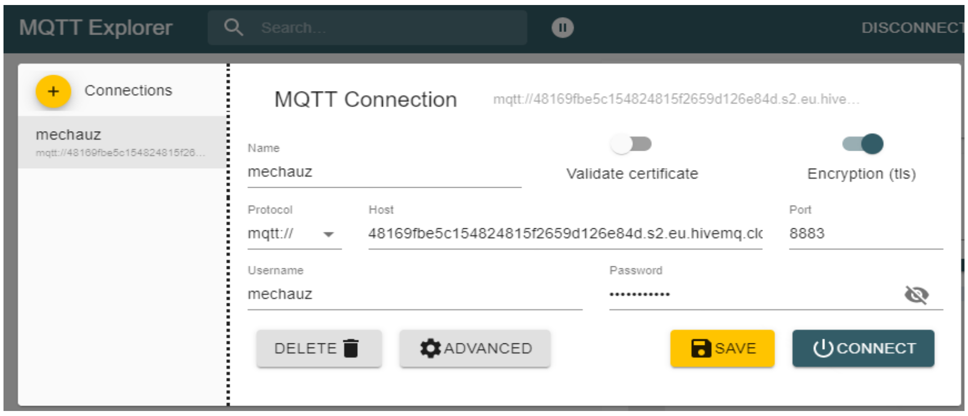
```
//////////////////////////////////////////////////////////////////////////////
// Handle incomming messages from the broker
//////////////////////////////////////////////////////////////////////////////
void MQTT_callback(char* topic, byte* payload, unsigned int length) {
  String response;
  for (int i = 0; i < length; i++) {
    response += (char)payload[i];
  }
  Serial.print("MQTT Message Arrived [");
  Serial.print(topic);
  Serial.print("] ");
  Serial.println(response);
  }
```

11. Make a copy of the sketch tested earlier and rename it to ***FP4_EX11_MQTT_client_push.ino*** Using the text editor of the Arduino IDE, adapt the code to change the state of a LED of another device from the button of yours. For this purpose, consider the following pairings:

GROUP #	PUBLISH	SUBSCRIBE
1	group01/btn_state group01/led_state	group02/btn_state group02/led_state
2	group02/btn_state group02/led_state	group01/btn_state group01/led_state
3	group03/btn_state group03/led_state	group05/btn_state group05/led_state
4	group04/btn_state group04/led_state	group03/btn_state group03/led_state
5	group05/btn_state group05/led_state	group04/btn_state group04/led_state

12. To train home, you can use an online MQTT Broker.
    - List of online MQTT Brokers with free access can be found here .
    - For development I recommend the use of MQTT Explorer client.

    In the following figure the cliente has been configured to use HIVEMQ .

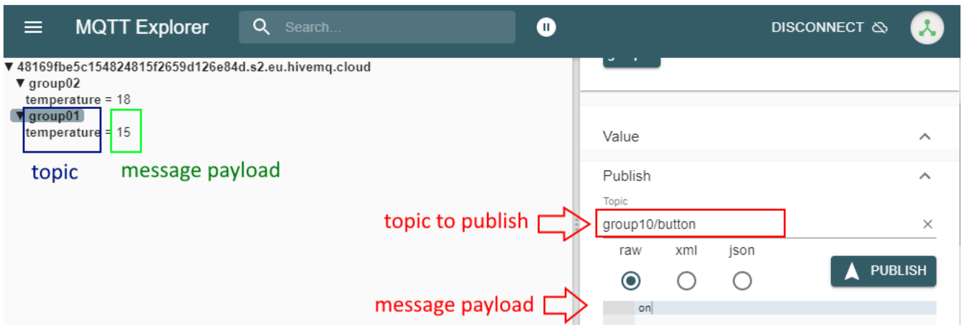


mechauz
mqtt://48169fbe5c154824815f26...

MQTT Connection   mqtt://48169fbe5c154824815f2659d126e84d.s2.eu.hive...

Name: mechauz
Validate certificate
Encryption (tls)
Protocol: mqtt://
Host: 48169fbe5c154824815f2659d126e84d.s2.eu.hivemq.clo
Port: 8883
Username: mechauz
Password: ••••••••••••

In the following example message ***"on"*** is published to topic group01/button



```
▼ 48169fbe5c154824815f2659d126e84d.s2.eu.hivemq.cloud
  ▼ group02
    temperature = 18
  ▼ group01
    temperature = 15
```

Topic: group10/button
[✓ raw] xml json
on