This page explains how to use ArduinoJson with [PubSubClient](https://github.com/knolleary/pubsubclient/), a popular MQTT library for Arduino. It shows how to use the JSON format in MQTT messages, but you can easily adapt the examples to use MessagePack instead.

Deserializing a JSON document in MQTT message

Once your program has subscribed to an MQTT topic, you can call [deserializeJson()](https://arduinojson.org/v6/api/json/deserializejson/) from the callback function.

void callback(char\* topic, byte\* payload, unsigned int length) {

StaticJsonDocument<256> doc;

deserializeJson(doc, payload, length);

// use the JsonDocument as usual...

}

Note that you don’t need to cast the payload pointer because [deserializeJson()](https://arduinojson.org/v6/api/json/deserializejson/) supports byte\* too.

Serializing a JSON document into an MQTT message

To publish a JSON document to an MQTT topic, you need to serialize it to a temporary buffer:

char buffer[256];

serializeJson(doc, buffer);

client.publish("outTopic", buffer);

You can save a few CPU cycles by passing the size of the payload to publish():

char buffer[256];

size\_t n = serializeJson(doc, buffer);

client.publish("outTopic", buffer, n);

# **ESP32: Sending JSON messages over MQTT**

[ESP32](https://techtutorialsx.com/category/esp32/) / [19 Comments](https://techtutorialsx.com/2017/04/29/esp32-sending-json-messages-over-mqtt/#comments)

The objective of this post is to explain how to send JSON messages over MQTT using the ESP32.

## ****Introduction****

The objective of this post is to explain how to send JSON messages over MQTT using the ESP32. To do so, we will be using two libraries that handle the low level details and expose us both the JSON encoding and the MQTT publishing functionalities in easy to use interfaces. These libraries are the [PubSubClient](https://github.com/knolleary/pubsubclient" \t "_blank), for the MQTT related functionality, and the [ArduinoJson](https://github.com/bblanchon/ArduinoJson" \t "_blank), for the JSON related functionality.

Both of the libraries work with the ESP8266 and the ESP32 and have some examples to help us getting started, which I encourage you to try. Also, I’ve been covering their use with both devices in previous posts, which are listed in the related posts section bellow.

As MQTT broker, we will use [CloudMQTT](https://www.cloudmqtt.com/" \t "_blank), which offers a free plan that allows us to create an account and test an instance of a broker. If you haven’t done it yet, please register and create an instance. We will need the instance information (**address**, **port**, **username** and **password**) latter.

## ****The code****

The first thing we need to do is including both the **ArduinoJson.h** and the **PubSubClient.h** libraries, in order to be able to access all the classes needed. We will also need to include the **WiFi.h** library, so we can connect to a WiFi network and thus be able to publish the messages to a MQTT topic.

In order to make the code more readable and easy to modify, we will declare some global variables to hold the credentials needed to connect to the WiFi network and to the MQTT broker. For the connection to the MQTT broker, we will need the information about the instance created, which is available in CloudMQTT instance information page.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | #include <ArduinoJson.h>  #include <WiFi.h>  #include <PubSubClient.h>    const char\* ssid = yourNetworkName";  const char\* password = "yourNetworkPassword";    const char\* mqttServer = "m11.cloudmqtt.com";  const int mqttPort = 12948;  const char\* mqttUser = "yourInstanceUsername";  const char\* mqttPassword = "yourInstancePassword"; |

After that, we will declare an object of class [WiFiClient](https://github.com/espressif/arduino-esp32/blob/master/libraries/WiFi/src/WiFiClient.h" \t "_blank). We will also declare an object of class [PubSubClient](https://github.com/knolleary/pubsubclient/blob/master/src/PubSubClient.cpp" \l "L17" \t "_blank), which receives as input of the constructor the previously declared WiFiClient instance. Note that we are not going to use the WiFiClient in our code, since it will be used by the PubSubClient library. So, we don’t need to worry about the implementation details.

|  |  |
| --- | --- |
| 1  2 | WiFiClient espClient;  PubSubClient client(espClient); |

After declaring the global variables, we will define the setup function. The first thing we do is opening a serial connection. Then, we connect to the WiFi network. Check [here](http://techtutorialsx.com/2017/04/24/esp32-connecting-to-a-wifi-network/) a detailed post which explains how to connect to a WiFi network with the ESP32.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | WiFi.begin(ssid, password);    while (WiFi.status() != WL\_CONNECTED) {    delay(500);    Serial.println("Connecting to WiFi..");  }    Serial.println("Connected to the WiFi network"); |

Next, we need to specify the address and the port of the MQTT broker we want to connect to.  To do so, we call the [setServer](https://github.com/knolleary/pubsubclient/blob/master/src/PubSubClient.cpp" \l "L565" \t "_blank) method on the PubSubClient object, passing as inputs the address and the port. These arguments were the ones specified as global variables and obtained from the CloudMQTT instance page information.

Now, we will connect to the MQTT server by calling the [connect](http://pubsubclient.knolleary.net/api.html#connect3) method. This method receives as input a string which corresponds to the unique identifier of the client (we will use “ESP32Client”). It also receives both the username and password needed to connect to the broker,  available at the CloudMQTT instance page information.

This method call will return true on connection success and false otherwise. We will do the connection attempt in a loop.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | while (!client.connected()) {      Serial.println("Connecting to MQTT...");        if (client.connect("ESP32Client", mqttUser, mqttPassword )) {          Serial.println("connected");        } else {          Serial.print("failed with state ");        Serial.print(client.state());        delay(2000);        }    } |

In the previous loop, we used two additional methods. The [connected](http://pubsubclient.knolleary.net/api.html#connected) method returns true if the connection is established or false if not. Thus, we use it as stopping condition for our connection loop.

The [state](http://pubsubclient.knolleary.net/api.html#state) method will return a code with information about why the connection failed. So, we use the value to print the reason of failure when it is not possible to connect to the broker.

After this, we should already be connected to the MQTT broker and we can start publishing messages to topics. Nevertheless, we will do this on the main loop function, and thus we don’t need more code on the setup function.

Now, in the main loop function, we declare an object of class [StaticJsonBuffer](https://github.com/bblanchon/ArduinoJson/blob/024976cda2fa9ea82e1f55504934b79ca53180c3/src/ArduinoJson/StaticJsonBuffer.hpp" \t "_blank), which will be used for creating the JSON message to be sent over MQTT. We will need to specify the capacity of the buffer in bytes as a template parameter. Then, we get a reference to a [JsonObject](https://bblanchon.github.io/ArduinoJson/api/jsonobject/constructor/" \t "_blank) from the StaticJsonBuffer object we created, by calling the [createObject](https://bblanchon.github.io/ArduinoJson/api/jsonbuffer/createobject/" \t "_blank) method.

|  |  |
| --- | --- |
| 1  2 | StaticJsonBuffer<300> JSONbuffer;  JsonObject& JSONencoder = JSONbuffer.createObject(); |

We will create a JSON message as indicated bellow. Note that we will create a message with static content just for keeping the code simple, but we could obtain the values from a sensor or other dynamic source.

|  |  |
| --- | --- |
| 1  2  3  4  5 | {  "device": "ESP32",  "sensorType" : "Temperature",  "Value" : [20,21,23]  } |

Next, we add the values to our JsonObject. You can check in more detail in [this](http://techtutorialsx.com/2017/04/27/esp32-creating-json-message/) previous post how to do it.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | JSONencoder["device"] = "ESP32";  JSONencoder["sensorType"] = "Temperature";  JsonArray& values = JSONencoder.createNestedArray("values");    values.add(20);  values.add(21);  values.add(23); |

Now, we will print the JSON message to a char buffer by using the [printTo](https://bblanchon.github.io/ArduinoJson/api/jsonarray/printto/" \t "_blank) method on the JsonObject reference. This method prints the message in a compact format, so it has the less overhead possible.

|  |  |
| --- | --- |
| 1  2 | char JSONmessageBuffer[100];  JSONencoder.printTo(JSONmessageBuffer, sizeof(JSONmessageBuffer)); |

Next, we send the JSON message to a topic called “**esp/test**“. We do it by calling the [publish](http://pubsubclient.knolleary.net/api.html#publish1) method on our PubSubClient object. This method will receive as inputs both the topic name and the message we want to publish.

As can be seen [here](http://pubsubclient.knolleary.net/api.html#publish1), this method returns “true” if the publish is successful and “false” otherwise. We will use this information for error handling.

Finally, we call the loop method on the same object reference. This should be called regularly so our program can process incoming messages and maintain the connection to the MQTT server. Since we are going to be sending the messages in the Arduino main loop function, with a small delay between each message, we call the loop method at the end of each iteration.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | Serial.println(JSONmessageBuffer);    if (client.publish("esp/test", JSONmessageBuffer) == true) {      Serial.println("Success sending message");  } else {      Serial.println("Error sending message");  }    client.loop(); |

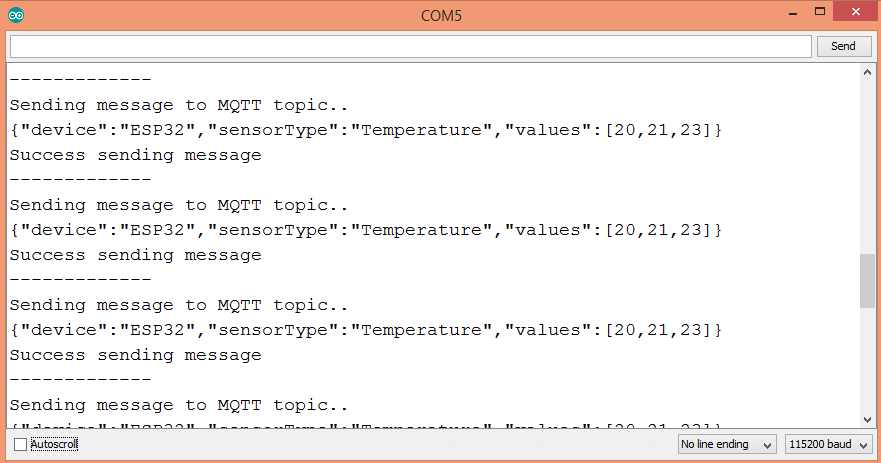
You can check final source code bellow, with some additional prints to help debugging.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78 | #include <ArduinoJson.h>  #include <WiFi.h>  #include <PubSubClient.h>    const char\* ssid = "yourNetworkName";  const char\* password =  "yourNetworkPassword";  const char\* mqttServer = "m11.cloudmqtt.com";  const int mqttPort = 12948;  const char\* mqttUser = "yourInstanceUsername";  const char\* mqttPassword = "yourInstancePassword";    WiFiClient espClient;  PubSubClient client(espClient);    void setup() {      Serial.begin(115200);    Serial.println();      WiFi.begin(ssid, password);      while (WiFi.status() != WL\_CONNECTED) {      delay(500);      Serial.println("Connecting to WiFi..");    }      Serial.println("Connected to the WiFi network");      client.setServer(mqttServer, mqttPort);      while (!client.connected()) {      Serial.println("Connecting to MQTT...");        if (client.connect("ESP32Client", mqttUser, mqttPassword )) {          Serial.println("connected");        } else {          Serial.print("failed with state ");        Serial.print(client.state());        delay(2000);        }    }    }    void loop() {      StaticJsonBuffer<300> JSONbuffer;    JsonObject& JSONencoder = JSONbuffer.createObject();      JSONencoder["device"] = "ESP32";    JSONencoder["sensorType"] = "Temperature";    JsonArray& values = JSONencoder.createNestedArray("values");      values.add(20);    values.add(21);    values.add(23);      char JSONmessageBuffer[100];    JSONencoder.printTo(JSONmessageBuffer, sizeof(JSONmessageBuffer));    Serial.println("Sending message to MQTT topic..");    Serial.println(JSONmessageBuffer);      if (client.publish("esp/test", JSONmessageBuffer) == true) {      Serial.println("Success sending message");    } else {      Serial.println("Error sending message");    }      client.loop();    Serial.println("-------------");      delay(10000);    } |

## ****Testing the code****

To test the code, we will use [MQTTLens](https://chrome.google.com/webstore/detail/mqttlens/hemojaaeigabkbcookmlgmdigohjobjm" \t "_blank) to subscribe to the topic where the ESP32 will be publishing. So, open the application and subscribe to the “**esp/test**” topic.

After that, upload the code to the ESP32 and open the serial console. You should get an output similar to figure 1.



**Figure 1** – Output of the program.

Now, go back to MQTTLens. You should get an output similar to figure 2, with the JSON messages being printed. MQTTLens recognizes the JSON format and, although we sent the message in a compact format, it allows us to see it in an indented user friendly format.

## Deserializing a JSON document in MQTT message

Once your program has received an MQTT message, you can call [deserializeJson()](https://arduinojson.org/v6/api/json/deserializejson/) and pass the MqttClient instance:

int messageSize = mqttClient.parseMessage();

if (messageSize) {

StaticJsonDocument<256> doc;

deserializeJson(doc, mqttClient);

// use the JsonDocument as usual...

}

This code leverages the [Stream](https://www.arduino.cc/reference/en/language/functions/communication/stream/) interface implemented by MqttClient, that’s why you don’t need to use a temporary buffer.

You can also use the same technique if you use a callback with MqttClient:

void onMqttMessage(int messageSize) {

StaticJsonDocument<256> doc;

deserializeJson(doc, mqttClient);

// use the JsonDocument as usual...

}

## Serializing a JSON document into an MQTT message

To publish a JSON document to an MQTT topic, you can use the [Stream](https://www.arduino.cc/reference/en/language/functions/communication/stream/) interface or use a temporary buffer.

To use the [Stream](https://www.arduino.cc/reference/en/language/functions/communication/stream/) interface, simply pass MqttClient to [serializeJson()](https://arduinojson.org/v6/api/json/serializejson/):

mqttClient.beginMessage(topic);

serializeJson(doc, mqttClient);

mqttClient.endMessage();

If you want to specify the size to beginMessage(), you could use [measureJson()](https://arduinojson.org/v6/api/json/measurejson/), but it’s faster to use a temporary buffer, like that:

char payload[64];

size\_t payloadSize = serializeJson(doc, payload);

mqttClient.beginMessage(topic, payloadSize);

mqttClient.print(payload);

mqttClient.endMessage();

//---------Java Json

<!DOCTYPE html>

<html>

<body>

<h2>Create JSON string from a JavaScript object.</h2>

<p id="demo"></p>

<p id="demo1"></p>

<p id="demo2"></p>

<p id="demo3"></p>

<script>

var obj = { name: "John", age: 30, city: "New York" };

var obj1 = {"DEN":["ON","OFF","ON","ON","ON","ON","ON","ON"],"wifiSignal":"82"}

var myJSON = JSON.stringify(obj1);

document.getElementById("demo").innerHTML = myJSON;

document.getElementById("demo1").innerHTML = obj1.wifiSignal;

var url=["OFF","ON","OFF","OFF","OFF","OFF","OFF","OFF"] ;

var SenData={"DEN": ["OFF","OFF","OFF","OFF","OFF","OFF","OFF","OFF"]};

for (var i=0;i<8;i++){

SenData.DEN[i]=url[i]; // the same add data into JSON DOccument {"DEN": ["OFF","OFF","OFF","OFF","OFF","OFF","OFF","OFF"]}

}

var myJSON2 = JSON.stringify(SenData);

document.getElementById("demo2").innerHTML = myJSON2;

document.getElementById("demo3").innerHTML = SenData.DEN[1];

</script>

</body>

</html>