

# IOT\_project3

---

**Biscini matteo [10709075]**

## INDEX:

1. [Requirements summary](#)
2. [Implementation](#)
  - [Data generation](#)
  - [Data receiving and message parsing](#)
  - [Message reaction](#)
3. [Testing](#)

## Requirements summary

Use node-red to parse a sequence of MQTT messages (saved in [challenge3.csv](#)) and perform different actions based on the message's content. In particular:

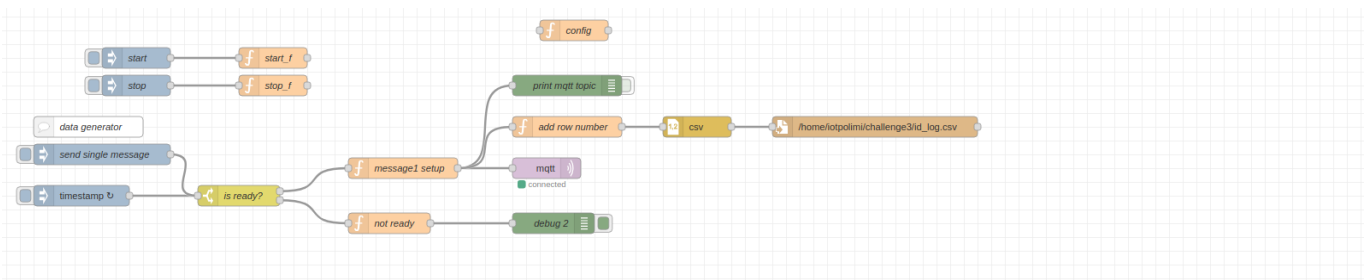
- if a message contains "Publish Message" the node will forward the message on the specified MQTT topic and the message payload is saved inside [filtered\\_pubs.csv](#).
- if the message contains an MQTT ACK the node will increment a global ack counter, perform an HTTP request on [ThingSpeak](#) update the field1 value with the ack number, the message content is saved inside [ack\\_log.csv](#).
- other messages will be discarded.

## Implementation

We will split the whole implementation into 3 different phases: data generation, data receiving and message parsing, message reaction.

**note:** the complete implementation is provided [here](#)

### Data generation



In this phase the flow will generate random data, it will save it in a CSV file and send it locally through MQTT on the "challenge3/id\_generator" topic with the following payload. If the inject named "timestamp" is enabled, a message every 5 seconds will be sent. All the messages sent in this phase are saved in a CSV file, named [id\\_log.csv](#).

```
{"id": 7781, "timestamp":1710930219} //message payload example
```

"Is ready?" switch is used, combined with start and stop to emulate a physical switch on the device.

## Relevant JS function & blocks

- **config (on start)**

It initializes all global variables to the desired values.

```
// Code added here will be run once
// whenever the node is started.
global.set("mqttDefaulChannel", "challenge3/id_generator");
global.set("ACKCounter", 0);
global.set("receivedMessagesCounter", 0);
global.set("tempCounter", 0);
global.set("thingSpeakKey", "VI5VOWUDI8Z5GCF1")
```

- **message1 setup (on message)**

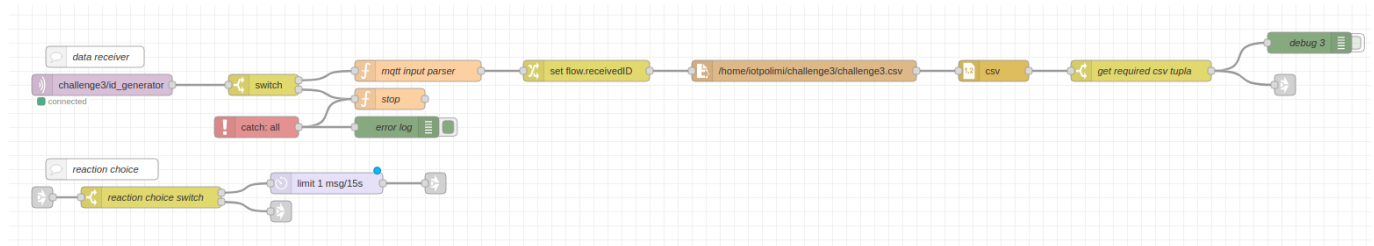
It prepares the MQTT message.

```
let mqtt_topic = String(global.get("mqttDefaulChannel")); //get topic
let jsonMessage = { "id": Math.random() * 5000, "timestamp": msg.payload};
//generate random paylod for the mqtt msg

msg.topic = mqtt_topic;
msg.payload = jsonMessage;

return msg;
```

## Data receiving and message parsing



This block will receive up to 80 messages on "challenge3/id\_generator" topic, and it uses the message id to get the correct row in the [challenge3.csv](#) file; based on the row content, the flow will be reacting differently (as specified in requirement summary). In this phase the correct reaction is triggered.

When the flow has already received 80 messages or occurs in error (in every stage), the stop function sets "is ready?" to false, stopping messages publishing on the "challenge3/id\_generator" topic.

### Relevant JS function & blocks

- **mqtt input parser (on message)**

It parses the received MQTT message.

```
global.set("receivedMessagesCounter",
(global.get("receivedMessagesCounter")+1)); //increment the message counter
msg.payload = parseInt(msg.payload.id % 7711); //compute the row number
return msg;
```

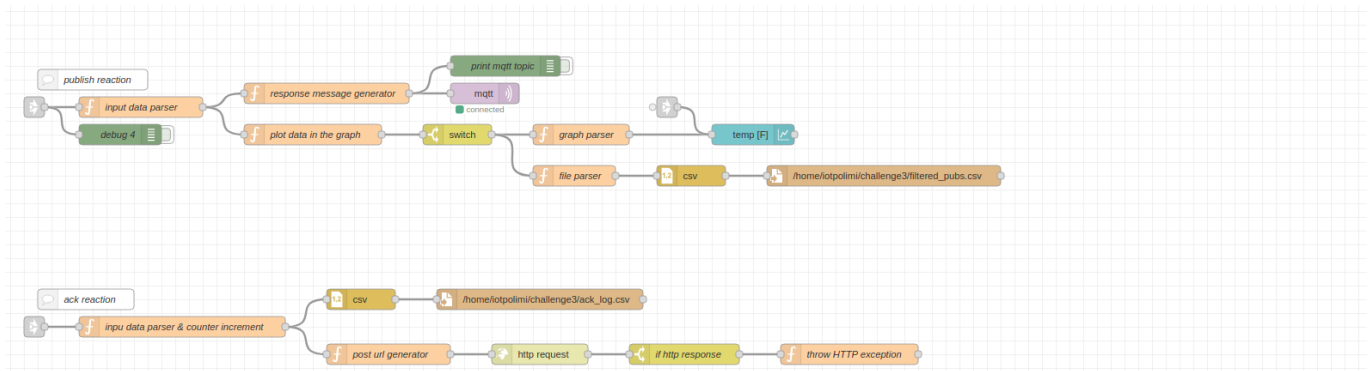
- **get required csv tuple (on message)**

It selects the correct row from the CSV file using the received id previously saved as a flow variable.

- **reaction choice switch**

Based on the received message, if required it triggers the correct reaction.

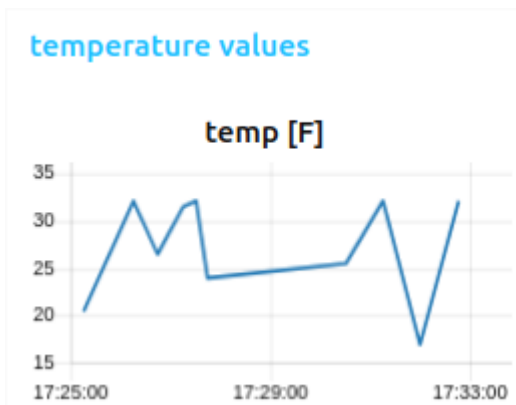
## Message reaction



The requirements ask to react to 2 different message classes: published and ACK messages.

### Publish reaction

In this phase the flow will react to MQTT messages of class "Publish Message", in particular publishing messages with specified payload on the required MQTT topic. Additionally, if a message payload contains a temperature in Fahrenheit, it is saved in the [filtered\\_pubs.csv](#) file and its value is plotted on a UI graph (a screenshot of this graph is provided below).



As shown in the following example, a single message can contain multiple MQTT topics and payload. In this terms the "input data parser" function block has the role of splitting the input message (a single string) in two arrays, containing respectively the topics and the payload of all the messages.

```
// received info & payload example, containing all the message payload and top
where forward
"Publish Message [hospital/room2], Publish Message [hospital/building5], Publish
Message [hospital/department2],"{"range": [4, 50], "description": "Room
Temperature", "type": "temperature", "unit": "C", "lat": 66, "long":
92},{ "type": "temperature", "lat": 81, "long": 95, "unit": "C",
"range": [3, 50], "description": "Room Temperature"}, {"description":
"Room Temperature", "lat": 55, "unit": "K", "type": "temperature",
"long": 88, "range": [7, 41]},"
```

## Relevant JS function & blocks

- **input data parser**

This block has to parse the input string into 2 arrays of JSON objects.

```
msg.payload.col9 = String(msg.payload.col9).split(','); //parse all the
MQTT topics and split them inside an array

if (msg.payload.col10) msg.payload.col10 = JSON.parse "[" +
msg.payload.col10 + "]"); //parse all the MQTT payloads and split them
inside an array
return msg;
```

- **response messages generator**

This block receives as input the 2 lists containing the topics and the payloads in a raw form and it has to translate it into the required format for the MQTT message.

```
let msgList = [];

let mqtt_topic = msg.payload.col9;
let id = msg.payload.col1;
let mqtt_payload = msg.payload.col10;

if(!mqtt_topic)mqtt_topic = global.get("mqttDefaultChannel");

mqtt_topic.forEach((element, i) => { //iterate on all the message topics
and payloads

    element = element.replace("Publish Message [", ''); //remove useless
parts from the message topic
    element = element.replace("]", '');
    element = element.replace(" ", '');

    let jsonMessage = { //prepare the message payload to send
        "timestamp": Date.now(),
        "id": id
    };

    if (mqtt_payload) { //if the received message has a payload add it to
the new message
        jsonMessage.payload = mqtt_payload[i];
    }

    let tmp = { //prepare the output object
        topic : element,
        payload : JSON.stringify(jsonMessage)
    }
    msgList.push(tmp); //add the output object to the output list
});
```

```
return [msgList];
```

- **plot data in the graph**

This block receives as input the list containing the payloads in raw form and has to compute the new value for the temperature graph; additionally, this block will prepare the pass condition (if the message contains temperature data with Fahrenheit as the measuring unit) for the following switch block.

```
let msgList = [];

let mqtt_payload = msg.payload.col10;

if (mqtt_payload) mqtt_payload.forEach((element) => {    //iterate on all
the message payloads

    let jsonMessage = {
        "display": +((element.type == "temperature") && (element.unit ==
"F")), //evaluate the pass condition, "display" will be equal to one of the
flow have to plot the data on the graph
        "value": (element.range[0] + element.range[1])/2,    //compute the
graph value
        "original_payload": element    //copy the original payload is
necessary to save it in the CSV file
    };

    let tmp = {    //prepare the output object
        payload: jsonMessage
    }
    msgList.push(tmp);    //add the output object to the output list
});

return [msgList];
```

- **graph parser**

This block gets the input message value discarding all the other input data not useful anymore.

- **file parser**

This block translates the original message payload in raw form, copied before, in the correct format required for the CSV file.

## Ack reaction

In this phase the flow will react to MQTT messages containing an ACK (of any type). ACK messages are saved on [ack\\_log.csv](#) file in terms of timestamp, sub\_id, msg\_type. Additionally, the flow will count the total amount of ACK messages (on a global counter) and it will publish that data on [ThingSpeak](#).

## Relevant JS function & blocks

- **input data parser & counter increment**

This block receives as input the original ACK message and has to translate it in the required format to save the ACK in the CSV file; additionally, it will increment the global ACK counter.

```
global.set("ACKCounter", (global.get("ACKCounter")+1)); //counter increment

let data = {           //data format translate
  timestamp : Date.now(),
  sub_id: msg.payload.col1,
  msg_type: msg.payload.col9,
}

msg.payload = data;

return msg;
```

- **post url generator**

This block will dynamically generate the URL required to perform the HTTP get request to the thingspeak API.

```
msg.url = "https://api.thingspeak.com/update?api_key=" +
global.get("thingSpeakKey") + "&field1=" + global.get("ACKCounter")
return msg;
```

- **throw HTTP exception**

The thingspeak API will respond with zero in case of problems with the HTTP get request, if such value is received this block will raise an exception consequently stopping the whole flow.

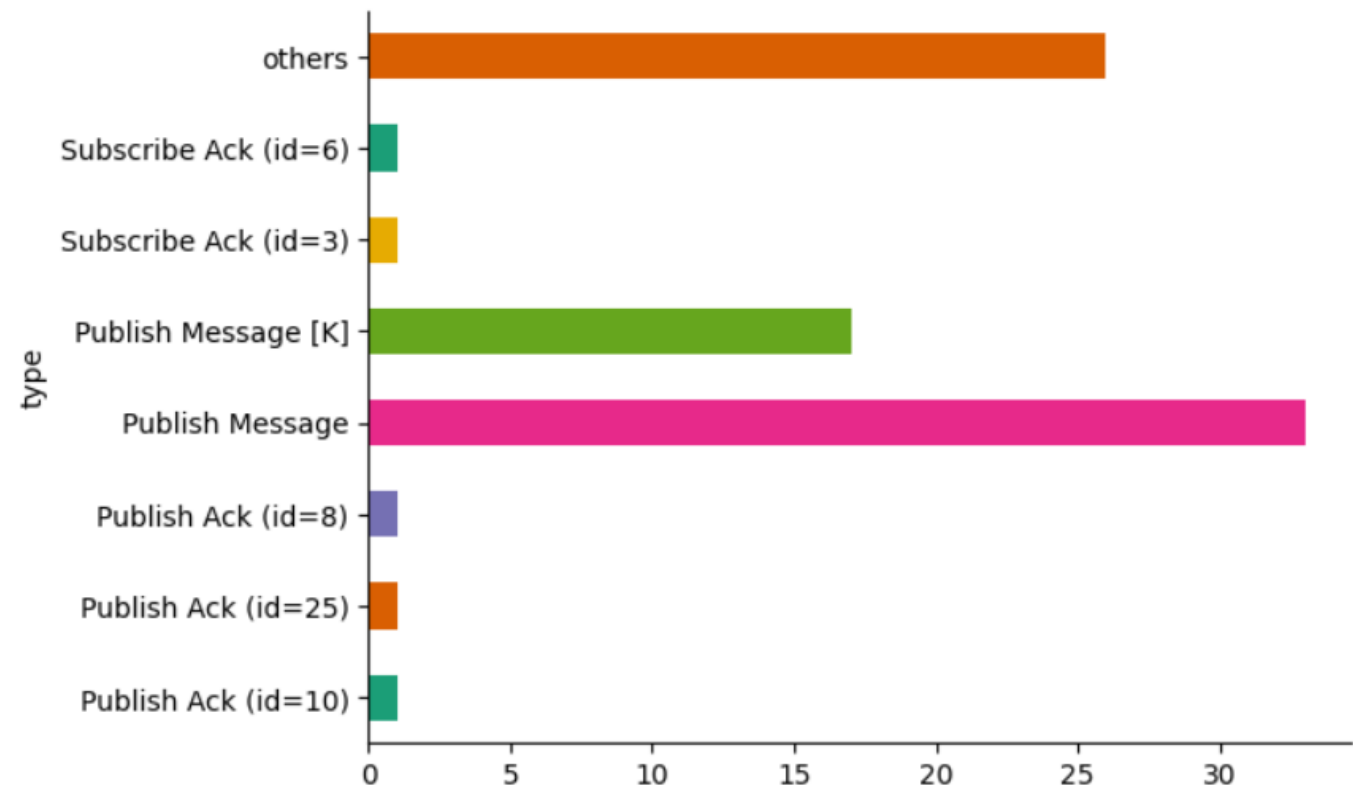


# Testing





























A Python tool for testing purposes is provided [here](#). The purpose of the tester is to verify the correct correlation between the various CSV files published from the flow during a run.































**note:** to generate a CSV files useful for testing, it is necessary to disable the limit on messages per second for the "publish reaction" brach, otherwise some messages can be discarded. Some CSV files generated in such way are provided in this [folder](#).























Following is provided the full test output:



Entire output

index	id	type	response
0	989	Publish Ack (id=25)	
1	1008	others	
2	845	Publish Message [K]	
3	2964	Publish Message	
4	1154	Publish Message [K]	
5	2529	Publish Message [K]	
6	446	Publish Message	
7	3362	others	
8	4807	Publish Message	
9	154	others	
10	3765	Publish Message	
11	1565	Publish Message [K]	
12	3464	Publish Message [K]	
13	1536	others	
14	1632	Publish Message	
15	498	Publish Message	
16	2496	Publish Message	
17	4636	others	
18	2265	others	
19	399	Publish Message [K]	
20	2063	Publish Message	
21	3673	others	
22	1496	others	
23	1247	Publish Message	
24	3465	Publish Message [K]	
25	339	others	
26	1514	Publish Message [K]	
27	632	Publish Ack (id=10)	

index	id	type	response
28	4742	Publish Message [K]	
29	2693	Publish Message	
30	1244	others	
31	2093	Publish Message	
32	3773	Publish Message	
33	1434	Publish Message	
34	3709	Publish Message	
35	162	others	
36	2827	Publish Message [K]	
37	3762	others	
38	2275	Publish Message	
39	4015	Publish Message	
40	3349	Publish Message [K]	
41	15	others	
42	3567	Publish Message	
43	4883	others	
44	1907	others	
45	3529	Publish Message	
46	101	Subscribe Ack (id=6)	
47	2820	others	
48	238	others	
49	3845	Publish Message	
50	4179	others	
51	1509	others	
52	226	others	
53	649	Publish Message [K]	
54	1279	others	
55	4732	Publish Message	
56	623	Publish Ack (id=8)	
57	3397	Publish Message	

index	id	type	response
58	1940	others	
59	3873	Publish Message [K]	
60	654	Publish Message	
61	2956	Publish Message [K]	
62	4226	Publish Message	
63	2340	Publish Message	
64	4664	Publish Message [K]	
65	921	Publish Message	
66	3334	Publish Message	
67	1795	Publish Message	
68	3307	others	
69	4433	Publish Message [K]	
70	986	Publish Message [K]	
71	1190	Publish Message	
72	391	Publish Message	
73	3709	Publish Message	
74	3746	others	
75	1965	Publish Message	
76	1807	others	
77	3600	Publish Message	
78	3810	Publish Message	
79	118	Subscribe Ack (id=3)	
80	1623	others	