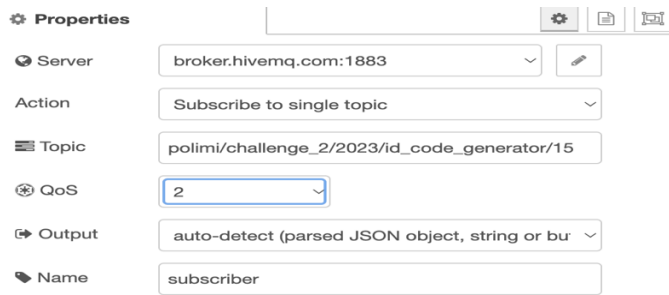


IoT 2023 CHALLENGE 2

First of all, we put **mqtt in** node to connect to the broker.hivemq.com:1883.



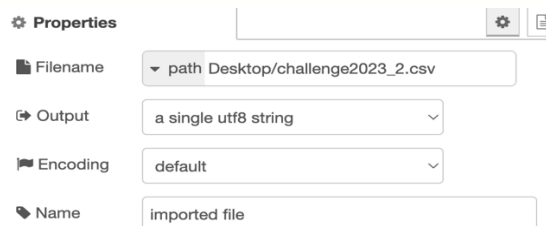
The screenshot shows the configuration for an MQTT In node. The 'Server' field is set to 'broker.hivemq.com:1883'. The 'Action' is 'Subscribe to single topic'. The 'Topic' is 'polimi/challenge_2/2023/id_code_generator/15'. The 'QoS' is set to '2'. The 'Output' is 'auto-detect (parsed JSON object, string or buffer)'. The 'Name' is 'subscriber'.

Then we put a function (frame number generator) to return the frame numbers that we need to compare with the 'No.' column in the CSV file. Here is the code that we used in this function block.

```
// defining the counter for limiting receiving just 100 messages
var counter = global.get('counter') || 0;
// saving the ID of the message for using it in the next nodes
var id = msg.payload.id;
global.set("msg_id", id);
// defining the frame_number based on our personal code.
//This value is because to search this value in CSV file
var frame_number = (parseInt(id) + 2693) % 7711;
global.set("frame_number", frame_number);
//Now increase the counter because the new message came, and then set it as a
global value.
counter++;
global.set('counter', counter);

msg.payload = frame_number;
//Here we are stopping to send messages after 100 messages
if (counter <= 100){
    return msg;
}
```

Then we are using the read file node to read the CSV file and we are using the below configuration.



The screenshot shows the configuration for a Read File node. The 'Filename' is set to 'path Desktop/challenge2023_2.csv'. The 'Output' is 'a single utf8 string'. The 'Encoding' is 'default'. The 'Name' is 'imported file'.

Then we are using the CSV node to convert the format of messages that we are receiving from the CSV file.

After it, we are using a function (CSV control). Below is the code that we used in this node to check if the message is published message or not. And then we check does the message has an empty payload or contains multiple messages. We send the string of messages to the next node.

```
//Get the frame_number and save it in a variable frame
var frame = global.get("frame_number");
//Check the frame number with 'No.' field of the CSV file.
if (frame == msg.payload['No.']){
    //Check if the message is published message or not based on "Info" column
    if (msg.payload.Info.includes('Publish Message')) {
        //Check if column "Message" does exist or not
        if (msg.payload.Message !== undefined){
            // save the content of the message in the payload and
return the msg

            var msg_content = msg.payload.Message;
            msg.payload = {};
            msg.payload = msg_content;
            return msg;
        }
        // return an empty message if "Message" column is not defined
        else {
            msg.payload = {};
            msg.payload = '{}';
            return msg;
        }
    }
}
```

After it, we used one split node to split the string by “}” and convert it to the array of messages. By using this node, we separate multiple messages to the array of messages.

Then we used another function (MSG_parser) to convert each separated message to the string format of JSON, and also add the ID and timestamp to the message.

```
var str;
// setting the msg_id and current timestamp
var id = global.get("msg_id");
var timestamp = Date.now();
// removing {} from the first and end of the string to handle
// the string in a simple way
msg.payload = msg.payload.replace('{', '');
msg.payload = msg.payload.replace('}', '');
// creating a string with the JSON format
//that contains "timestamp", "ID", "payload"
str = "{\"timestamp\": \"" + timestamp + "\", \"id\": \"" + id +
    "\", \"payload\": \"" + msg.payload + "\"}";
msg.payload = str;
return msg;
```

Then we send these separate messages to the broker with the below configuration.

⚙️

Properties

⚙️

📄

🖼️

🌐

Server

broker.hivemq.com:1883

✎

📋

Topic

/polimi/iot2023/challenge2/10832693

⚙️

QoS

0

🔄

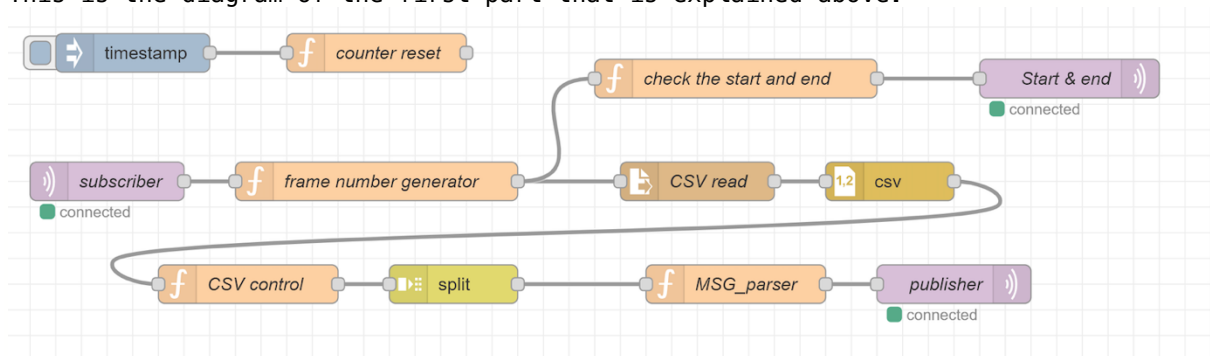
Retain

🏷️

Name

publisher

This is the diagram of the first part that is explained above.



For sending Start and END message at the beginning and finishing of the execution we put a function (check the start and end) to control the flow of the messages. while we received the first message, we are sending Start message. The below code is the way we handled it. We just checked if the counter is equal to 1 then send Start message. For sending END message we did the same procedure, but we checked the condition if the counter is equal to 100 then send an end message.

```

// send START message to the broker before starting the procedure
var counter = global.get("counter");
if(counter == 1){
    msg.payload = 'START';
    return msg;
}
// check if the message generation procedure finished then send
// END message to the broker
else if (counter == 100) {
    msg.payload = 'END';
    return msg;
}
  
```

And here is the configuration for publishing the start and end message to the same topic. The node "start & end" does this procedure.

Properties

Server

broker.hivemq.com:1883

Topic

/polimi/iot2023/challenge2/10832693

QoS

0

Retain

Name

Start & end

We did an additional thing for resetting the counter. Because sometimes we want to restart the message generation and by doing so we defined a timestamp and function(counter reset) to reset the counter to zero.

```
var counter = global.get("counter");
global.set("counter",0);
return msg;
```

Now the publishing part is finished, and we need to subscribe to the same broker that we created with the topic of our person code. Here are the configuration details.

Properties

Server

broker.hivemq.com:1883

Action

Subscribe to single topic

Topic

/polimi/iot2023/challenge2/10832693

QoS

2

Output

auto-detect (parsed JSON object, string or bu

Name

subscriber

Then we are putting a function (temp_generator) to extract the temperature that is in Celsius. For doing this we need to select the upper bound of the temperature.

```
/* receiving message from broker
and check the type of temperature if it is "C" */
if (msg.payload.payload.unit == 'C'){
    // select the upper bound of the temrature and return it as the payload
```

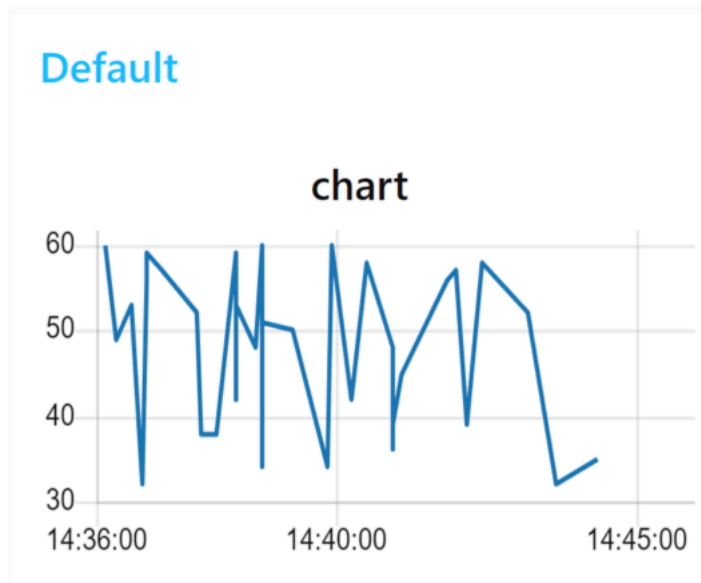
```

    msg.payload = msg.payload.payload.range[1];

    return msg;
}

```

After this, we are using the Chart node for plotting these extracted temperatures. And here is our chart.



Then we need to save these messages in a CSV file. For doing this we used a function (final output generator). Here is the function that send these messages to CSV node.

```

/* take the messages that are contain temperature in Celsius and
save them in CSV file */
if (msg.payload.payload.unit == 'C'){
    return msg;
}

```

And here are the configurations of how we can save the CSV file as we want.

Properties

Columns

id,timestamp,payload

Separator

comma

Name

Name

CSV to Object options

Input

Skip first lines

☐ first row contains column names
☒ parse numerical values
☒ include empty strings
☒ include null values

Output

a message per row

Object to CSV options

Output

send headers once, until msg.reset

