Internet Of Things (IOT) - Challenge 3

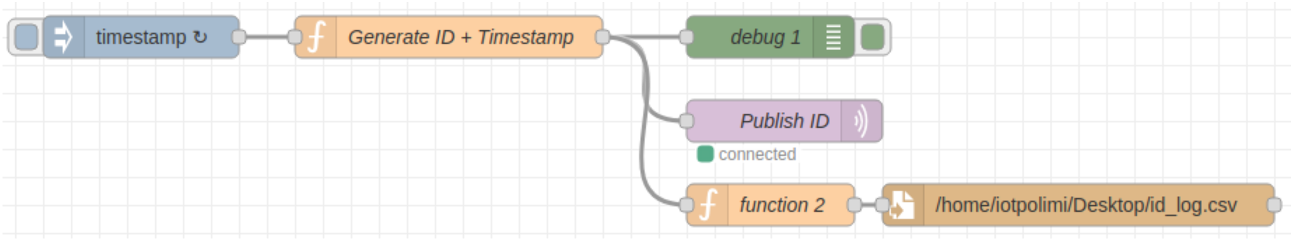
Mohammadreza Zamani (10869960) – Asal Abbasnejadfard (10974178)

Challenge: What to do? (1)

Create a flow to periodically publish MQTT messages to the local mosquitto broker (localhost, port 1884), to the topic *challenge3/id_generator* (be sure to start the mosquitto broker locally with the correct port)

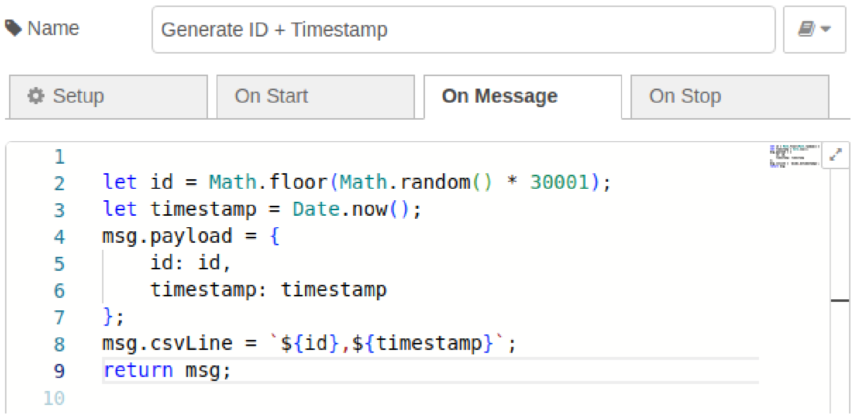
Solution:

To do this part we use the structure below in node-red.



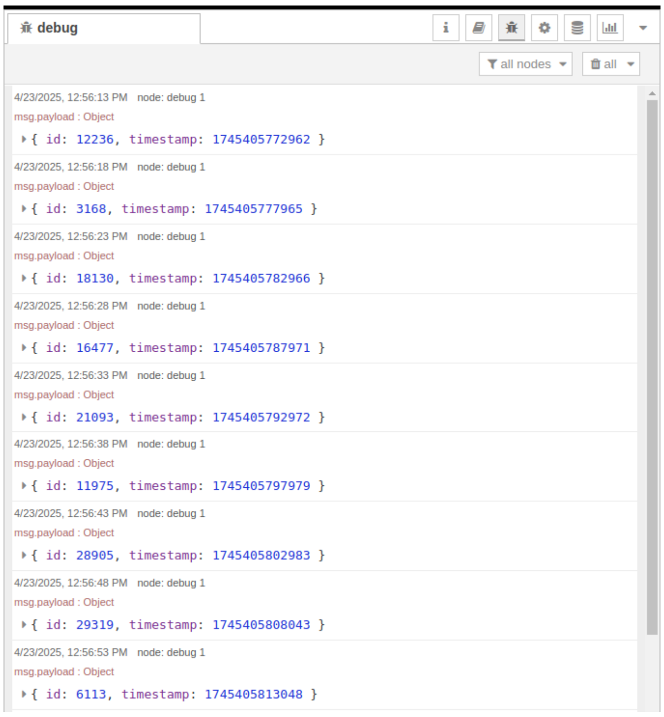
Inject node initiates the flow every 5 seconds. It simply passes the current timestamp to the next Generate ID + Timestamp function. This function generates a random number between 0 and 30000 as an id and the current UNIX timestamp. These values are stored in msg.payload as a JSON object. It also creates a CSV-formatted string (msg.csvLine) for later use.

This is the code of Generate ID + Timestamp function



```
let id = Math.floor(Math.random() * 30001);
let timestamp = Date.now();
msg.payload = {
    id: id,
    timestamp: timestamp
};
msg.csvLine = `${id},${timestamp}`;
return msg;
```

Then we use a debug node to verify the output structure of the msg.payload, ensuring the ID and timestamp are generated correctly. You can see some sample output from debug 1.

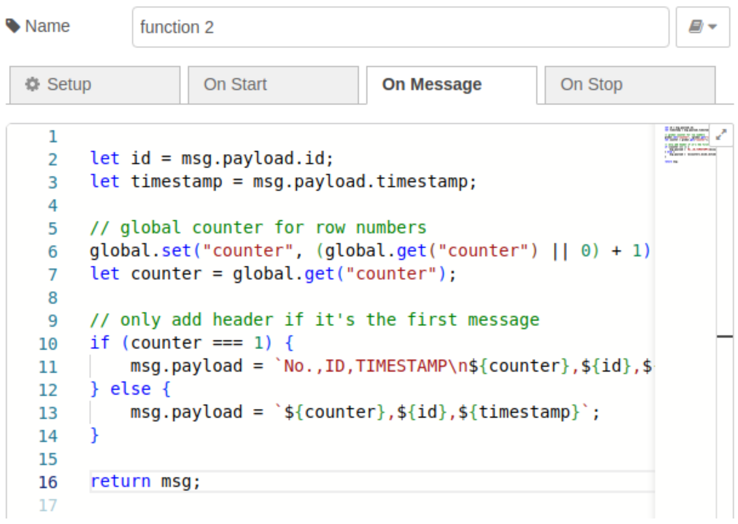


```
4/23/2025, 12:56:13 PM  node: debug 1
msg.payload : Object
▸{ id: 12236, timestamp: 1745405772962 }
4/23/2025, 12:56:18 PM  node: debug 1
msg.payload : Object
▸{ id: 3168, timestamp: 1745405777965 }
4/23/2025, 12:56:23 PM  node: debug 1
msg.payload : Object
▸{ id: 18130, timestamp: 1745405782966 }
4/23/2025, 12:56:28 PM  node: debug 1
msg.payload : Object
▸{ id: 16477, timestamp: 1745405787971 }
4/23/2025, 12:56:33 PM  node: debug 1
msg.payload : Object
▸{ id: 21093, timestamp: 1745405792972 }
4/23/2025, 12:56:38 PM  node: debug 1
msg.payload : Object
▸{ id: 11975, timestamp: 1745405797979 }
4/23/2025, 12:56:43 PM  node: debug 1
msg.payload : Object
▸{ id: 28905, timestamp: 1745405802983 }
4/23/2025, 12:56:48 PM  node: debug 1
msg.payload : Object
▸{ id: 29319, timestamp: 1745405808043 }
4/23/2025, 12:56:53 PM  node: debug 1
msg.payload : Object
▸{ id: 6113, timestamp: 1745405813048 }
```

Publish ID node Publishes the generated message (msg.payload) to the topic challenge3/id_generator on the local broker. Function 2 node This node formats the message into a CSV line. It also keeps track of a global counter (global.set("counter")) to write an incremental No. for each message. If the counter is 1, it adds the CSV header.

Finally, write file node Writes the final CSV string from function 2 into a file located at /home/iotpolimi/Desktop/id_log.csv. It appends new lines and does not overwrite the file. Also, we create **id_log.csv**.

This is the code of function 2



```

let id = msg.payload.id;
let timestamp = msg.payload.timestamp;

// global counter for row numbers
global.set("counter", (global.get("counter") || 0) + 1)
let counter = global.get("counter");

// only add header if it's the first message
if (counter === 1) {
    msg.payload = `No.,ID,TIMESTAMP\n${counter},${id},$
} else {
    msg.payload = `${counter},${id},${timestamp}`;
}

return msg;

```

Challenge: What to do? (2)

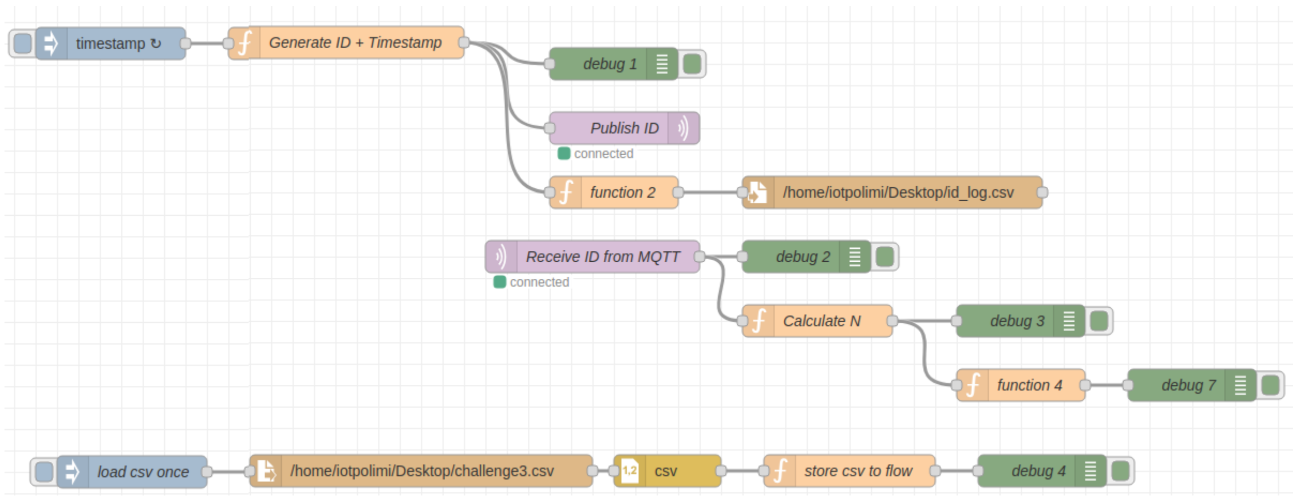
In another branch of the flow (same flow):

Subscribe to the topic *challenge3/id_generator* in the local broker (localhost, port 1884). After receiving a message from the subscription, take the ID and compute the remainder of the division by 7711 to get N:

N = ID modulo 7711

Solution:

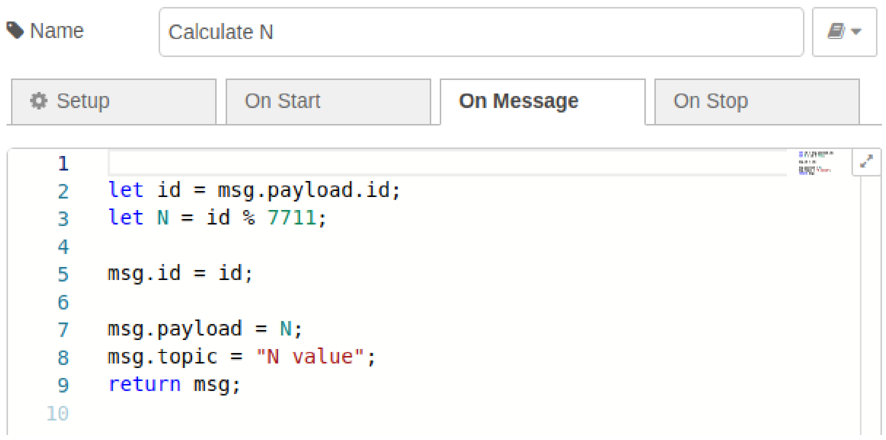
The structure of nodes and functions in this part are



We receive ID from Mqtt in. It receives the messages previously published in the previous part, each containing a JSON payload with an ID and timestamp. The debug 2, Logs the incoming message payload received from the MQTT subscription and helps verify that messages are properly received from the broker.

Calculate N function Extracts the id from the payload and computes N as the modulo 7711 of the ID (N = id % 7711).

This is the code of calculating N function

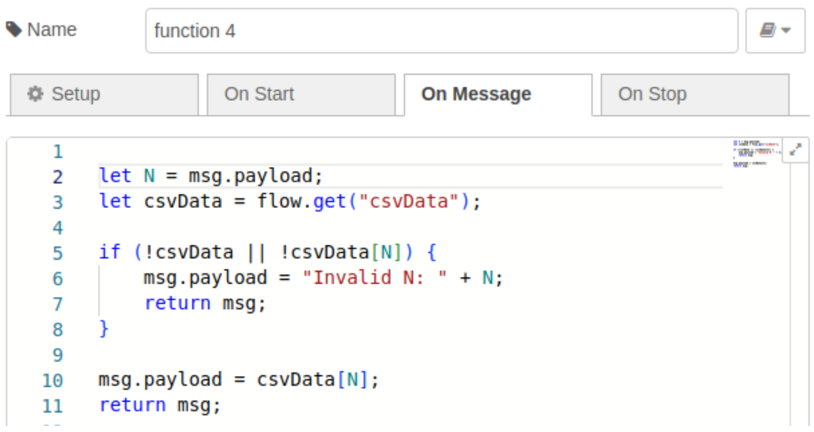


Name: Calculate N

```

let id = msg.payload.id;
let N = id % 7711;

msg.id = id;

msg.payload = N;
msg.topic = "N value";
return msg;

```

Debug 3, Logs the computed value of N and allows us to confirm that N is correctly calculated before using it. Function 4 looks up the row from the stored CSV data where the line number equals N and extracts the reference row corresponding to the computed N.

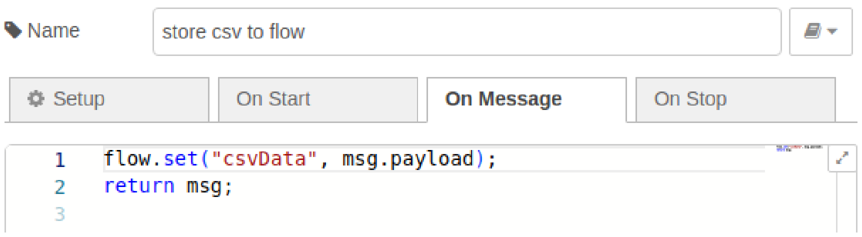
This is the code of function 4



```

let N = msg.payload;
let csvData = flow.get("csvData");

if (!csvData || !csvData[N]) {
    msg.payload = "Invalid N: " + N;
    return msg;
}

msg.payload = csvData[N];
return msg;
```

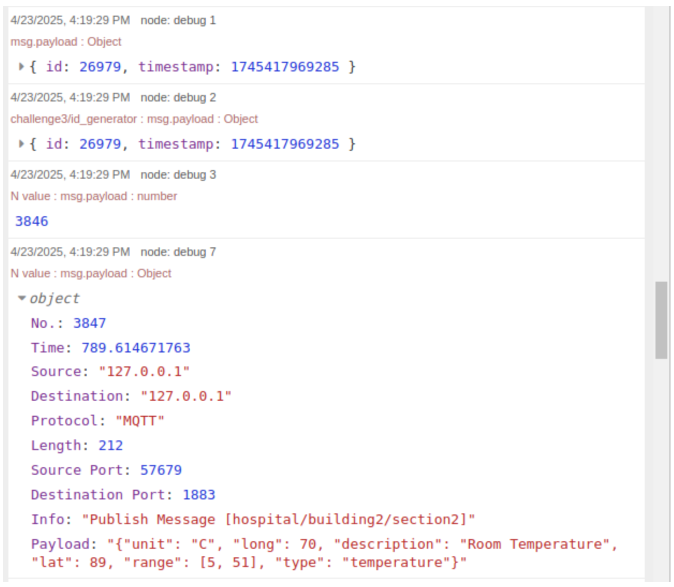
Debug 7 displays the extracted row corresponding to the N-th entry in challenge3.csv.
The load CSV once (inject). Triggers the reading of the file challenge3.csv once when clicked. File In Node loads rows of challenge3.csv into the Node-RED flow. CSV node Converts raw CSV lines into structured JSON format for easier data manipulation. The store CSV to flow function makes the CSV data accessible for other nodes and functions when needed.

This is the code of the store CSV flow function

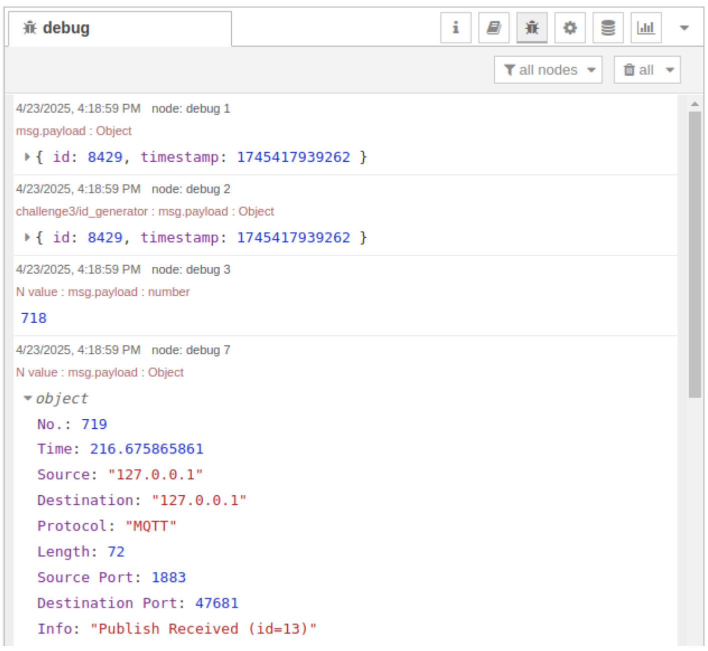


```
flow.set("csvData", msg.payload);
return msg;

```

There are some samples outputs of debug 1, debug 2, debug3 and debug 7.



```
4/23/2025, 4:19:29 PM   node: debug 1
msg.payload : Object
▸{ id: 26979, timestamp: 1745417969285 }
4/23/2025, 4:19:29 PM   node: debug 2
challenge3/id_generator : msg.payload : Object
▸{ id: 26979, timestamp: 1745417969285 }
4/23/2025, 4:19:29 PM   node: debug 3
N value : msg.payload : number
3846
4/23/2025, 4:19:29 PM   node: debug 7
N value : msg.payload : Object
▾object
  No.: 3847
  Time: 789.614671763
  Source: "127.0.0.1"
  Destination: "127.0.0.1"
  Protocol: "MQTT"
  Length: 212
  Source Port: 57679
  Destination Port: 1883
  Info: "Publish Message [hospital/building2/section2]"
  Payload: "{"unit": "C", "long": 70, "description": "Room Temperature",
  "lat": 89, "range": [5, 51], "type": "temperature"}"
```



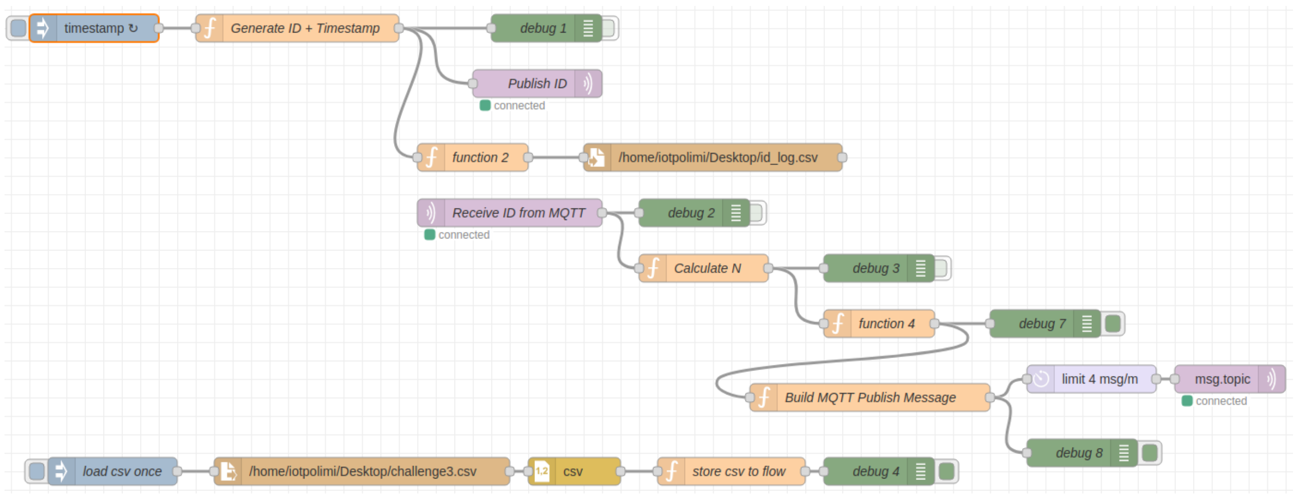
```
4/23/2025, 4:18:59 PM   node: debug 1
msg.payload : Object
▸{ id: 8429, timestamp: 1745417939262 }

4/23/2025, 4:18:59 PM   node: debug 2
challenge3/id_generator : msg.payload : Object
▸{ id: 8429, timestamp: 1745417939262 }

4/23/2025, 4:18:59 PM   node: debug 3
N value : msg.payload : number
718

4/23/2025, 4:18:59 PM   node: debug 7
N value : msg.payload : Object
▾object
  No.: 719
  Time: 216.675865861
  Source: "127.0.0.1"
  Destination: "127.0.0.1"
  Protocol: "MQTT"
  Length: 72
  Source Port: 1883
  Destination Port: 47681
  Info: "Publish Received (id=13)"
```

Challenge: What to do? (3)

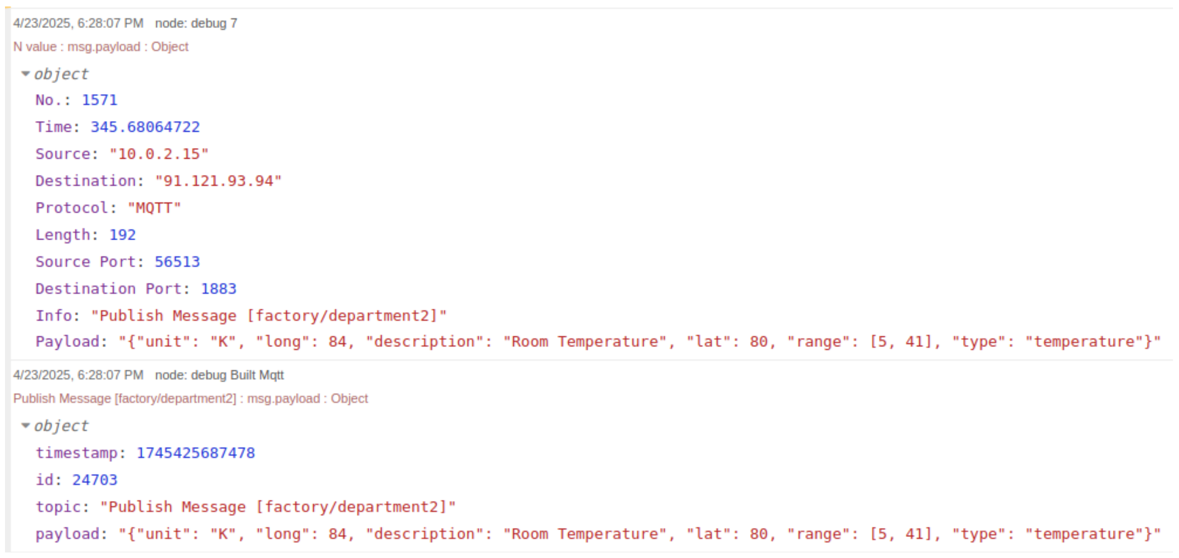
If the message with **Frame No. = N** in the file contains an **MQTT Publish** then, send a publish message to the local broker to the same topic found in the **MQTT Publish** The message you publish should have as payload the following string: ‘{"timestamp":"CURRENT_TIMESTAMP","id":"SUB_ID","topic":"MQTT_PUBLISH_TOPIC", "payload":"MQTT_PUBLISH_PAYLOAD"}’

Solution:

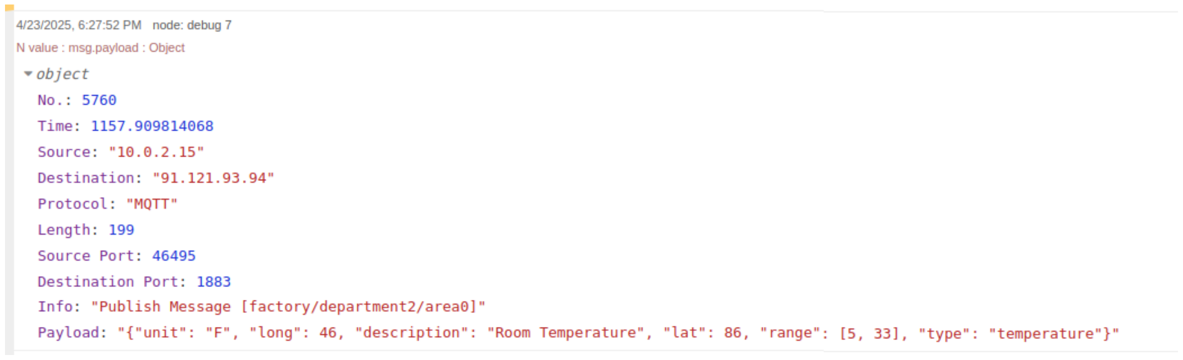
The flow of part 3



Build MQTT Publish Message function reads fields from the selected CSV row (specifically Info and Payload) and builds a new JSON message. Delay node ensures no more than 4 messages per minute are published (rate limiter). Mqtt out node Publishes the final built message to the topic defined by the CSV’s "Info" field (used as the MQTT topic). Debug Built Mqtt displays the final publishing message before it is sent via MQTT. There are some samples of debug 7 and debug Built Mqtt.



```
4/23/2025, 6:28:07 PM   node: debug 7
N value : msg.payload : Object
▾object
  No.: 1571
  Time: 345.68064722
  Source: "10.0.2.15"
  Destination: "91.121.93.94"
  Protocol: "MQTT"
  Length: 192
  Source Port: 56513
  Destination Port: 1883
  Info: "Publish Message [factory/department2]"
  Payload: "{"unit": "K", "long": 84, "description": "Room Temperature", "lat": 80, "range": [5, 41], "type": "temperature"}"
4/23/2025, 6:28:07 PM   node: debug Built Mqtt
Publish Message [factory/department2] : msg.payload : Object
▾object
  timestamp: 1745425687478
  id: 24703
  topic: "Publish Message [factory/department2]"
  payload: "{"unit": "K", "long": 84, "description": "Room Temperature", "lat": 80, "range": [5, 41], "type": "temperature"}"
```



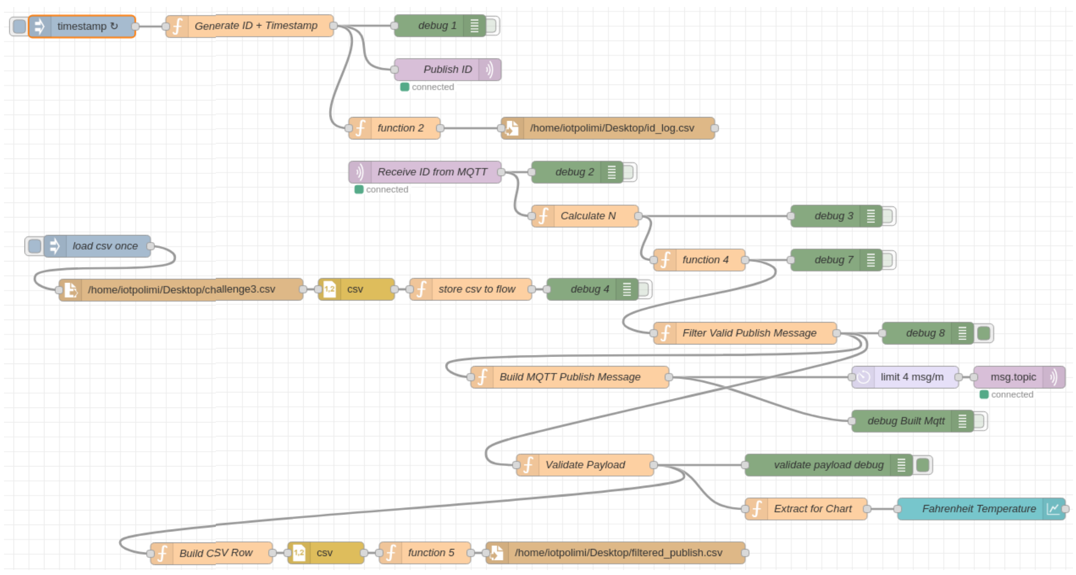
```
4/23/2025, 6:27:52 PM   node: debug 7
N value : msg.payload : Object
▾object
  No.: 5760
  Time: 1157.909814068
  Source: "10.0.2.15"
  Destination: "91.121.93.94"
  Protocol: "MQTT"
  Length: 199
  Source Port: 46495
  Destination Port: 1883
  Info: "Publish Message [factory/department2/area0]"
  Payload: "{"unit": "F", "long": 46, "description": "Room Temperature", "lat": 86, "range": [5, 33], "type": "temperature"}"
```

Challenge: What to do? (4)

In addition, after publishing the publish message, if the Publish Message contains in the payload a temperature in Fahrenheit (check for the Type=Temperature and Unit=F attributes in the payload), take this message and plot its value in a Node-Red chart.

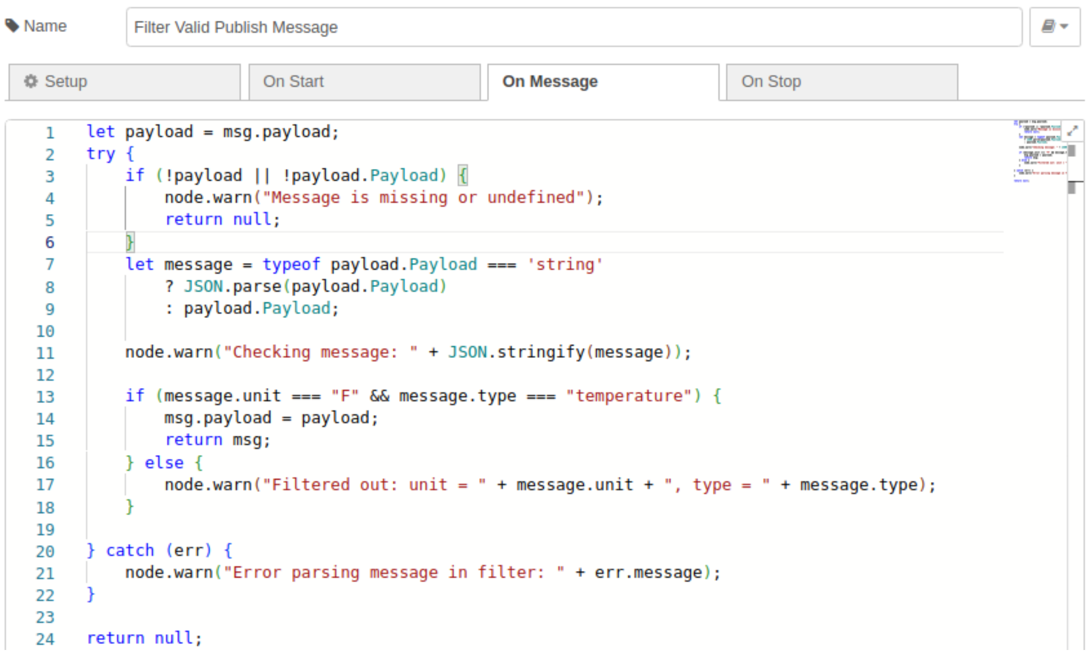
Solution:

This is the flow of this part



Filter valid publish message function filters out only those messages whose payloads include a temperature in Fahrenheit. It also ensures that the Payload field is present and correctly formatted as a JSON object. Only valid messages pass through for further processing.

This is the code of the Filter valid publish message function.



Name: Filter Valid Publish Message

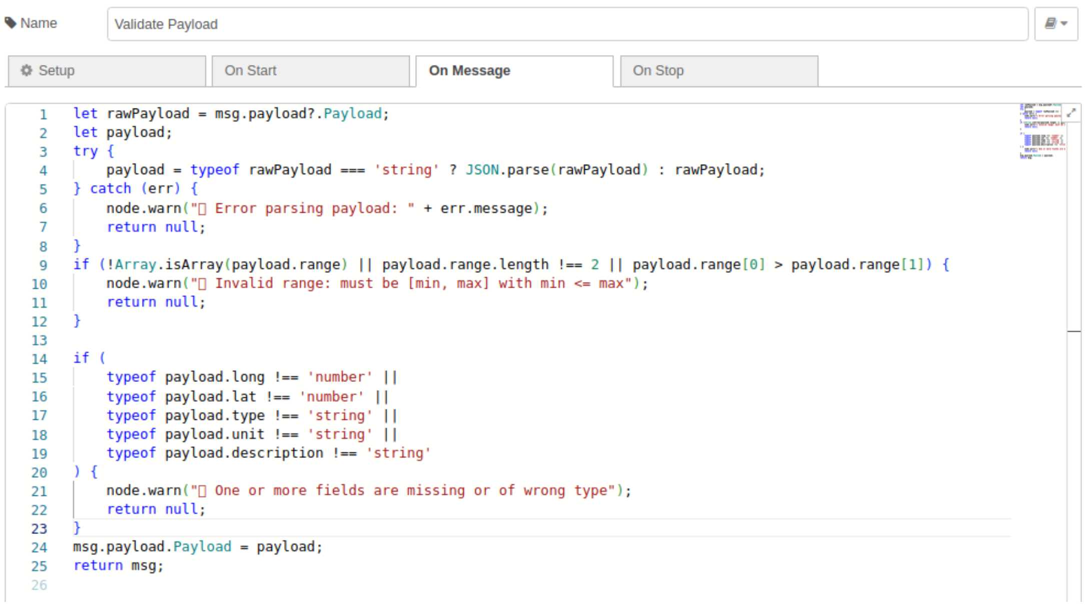
Setup | On Start | **On Message** | On Stop

```javascript
let payload = msg.payload;
try {
    if (!payload || !payload.Payload) {
        node.warn("Message is missing or undefined");
        return null;
    }
    let message = typeof payload.Payload === 'string'
        ? JSON.parse(payload.Payload)
        : payload.Payload;

    node.warn("Checking message: " + JSON.stringify(message));

    if (message.unit === "F" && message.type === "temperature") {
        msg.payload = payload;
        return msg;
    } else {
        node.warn("Filtered out: unit = " + message.unit + ", type = " + message.type);
    }

} catch (err) {
    node.warn("Error parsing message in filter: " + err.message);
}

return null;
```

Debug 8 is connected immediately after the Filter Valid Publish Message function. Its main role is to Monitor which messages passed the filtering condition (i.e., only temperature messages in Fahrenheit).

Validate payload function performs deep validation on the message's payload. For example, Ensures fields like long, lat, range, type, unit, and description exist and are valid. Confirms that range is an array of two numbers where the first ≤ second. Returns the cleaned and validated message.

This is the code of the Filter valid payload function.



Name: Validate Payload

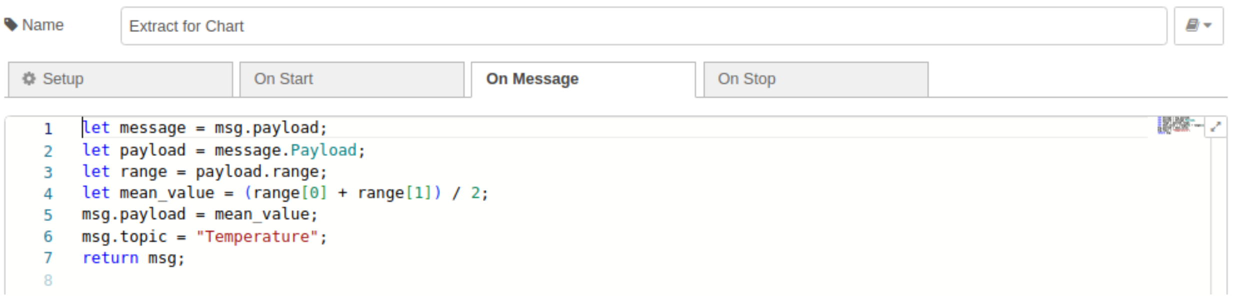
Setup | On Start | **On Message** | On Stop

```javascript
let rawPayload = msg.payload?.Payload;
let payload;
try {
    payload = typeof rawPayload === 'string' ? JSON.parse(rawPayload) : rawPayload;
} catch (err) {
    node.warn("□ Error parsing payload: " + err.message);
    return null;
}
if (!Array.isArray(payload.range) || payload.range.length !== 2 || payload.range[0] > payload.range[1]) {
    node.warn("□ Invalid range: must be [min, max] with min <= max");
    return null;
}

if (
    typeof payload.long !== 'number' ||
    typeof payload.lat !== 'number' ||
    typeof payload.type !== 'string' ||
    typeof payload.unit !== 'string' ||
    typeof payload.description !== 'string'
) {
    node.warn("□ One or more fields are missing or of wrong type");
    return null;
}
msg.payload.Payload = payload;
return msg;

```

validate payload debug node is connected to the output of the Validate Payload function.

Extract for Chart function extracts the mean temperature value from the range attribute of the payload.

This is the code of the Extract for chart function.

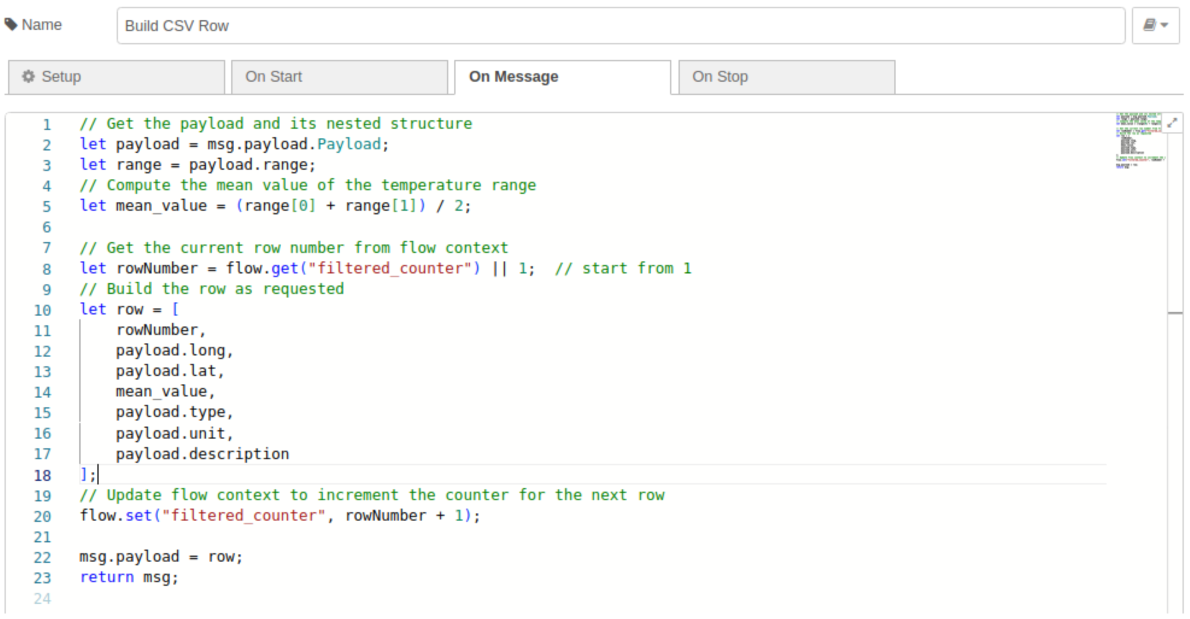


Name: Extract for Chart

```
let message = msg.payload;
let payload = message.Payload;
let range = payload.range;
let mean_value = (range[0] + range[1]) / 2;
msg.payload = mean_value;
msg.topic = "Temperature";
return msg;
```

Fahrenheit Temperature node, a line chart node that plots the extracted mean temperature values over time for all publish messages with Fahrenheit temperature. Build CSV row function generates a CSV row for each valid Fahrenheit message. It includes Incremental row number (No.), LONG, LAT, Mean temperature (MEAN_VALUE), TYPE, UNIT, and DESCRIPTION. Csv node converts the array format into a properly comma-separated CSV string.

This is the code of the Build CSV row function.

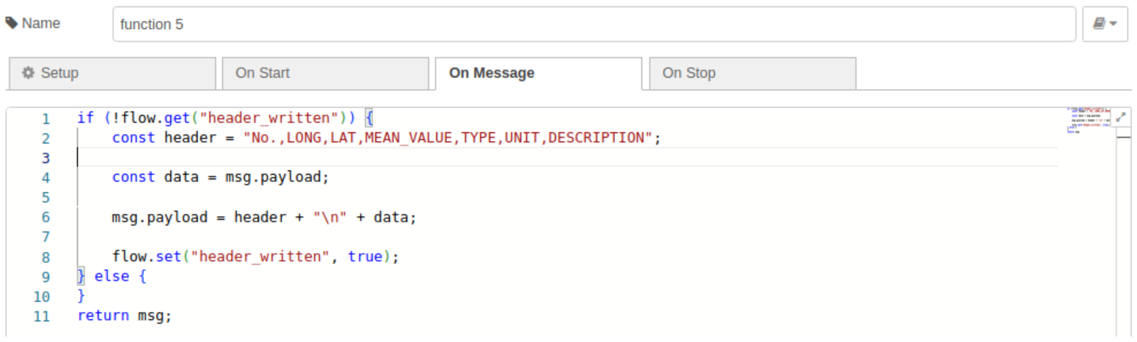


Name: Build CSV Row

```
// Get the payload and its nested structure
let payload = msg.payload.Payload;
let range = payload.range;
// Compute the mean value of the temperature range
let mean_value = (range[0] + range[1]) / 2;

// Get the current row number from flow context
let rowNumber = flow.get("filtered_counter") || 1;  // start from 1
// Build the row as requested
let row = [
    rowNumber,
    payload.long,
    payload.lat,
    mean_value,
    payload.type,
    payload.unit,
    payload.description
];
// Update flow context to increment the counter for the next row
flow.set("filtered_counter", rowNumber + 1);

msg.payload = row;
return msg;
```

Function 5, Ensures that the CSV header is written only once at the start of the file Checks a flow variable header written before writing the header and Appends the header only for the first row.

This is the code of function 5.

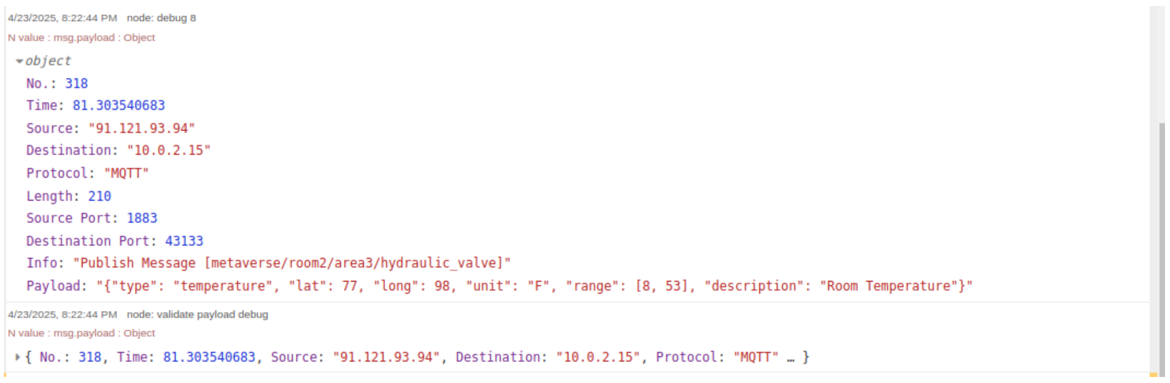


Name: function 5

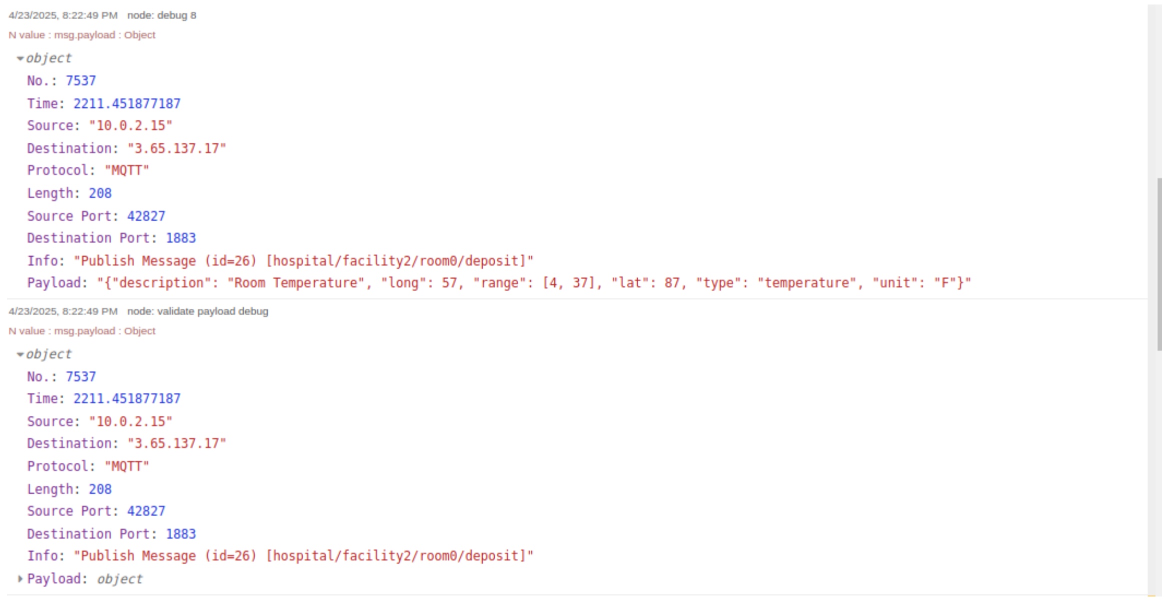
```
if (!flow.get("header_written")) {
    const header = "No.,LONG,LAT,MEAN_VALUE,TYPE,UNIT,DESCRIPTION";

    const data = msg.payload;

    msg.payload = header + "\n" + data;

    flow.set("header_written", true);
} else {
}
return msg;
```

Write file node Appends each CSV row (with header once) to a file named filtered_publish.csv. This file contains only Fahrenheit temperature publish messages, as required.

There are some samples of debug 8 and validate payload debug. Also, we create **filtered_publish.csv**.



```
4/23/2025, 8:22:44 PM   node: debug 8
N value : msg.payload : Object
▾object
  No.: 318
  Time: 81.303540683
  Source: "91.121.93.94"
  Destination: "10.0.2.15"
  Protocol: "MQTT"
  Length: 210
  Source Port: 1883
  Destination Port: 43133
  Info: "Publish Message [metaverse/room2/area3/hydraulic_valve]"
  Payload: "{"type": "temperature", "lat": 77, "long": 98, "unit": "F", "range": [8, 53], "description": "Room Temperature"}"
4/23/2025, 8:22:44 PM   node: validate payload debug
N value : msg.payload : Object
▸{ No.: 318, Time: 81.303540683, Source: "91.121.93.94", Destination: "10.0.2.15", Protocol: "MQTT" … }
```



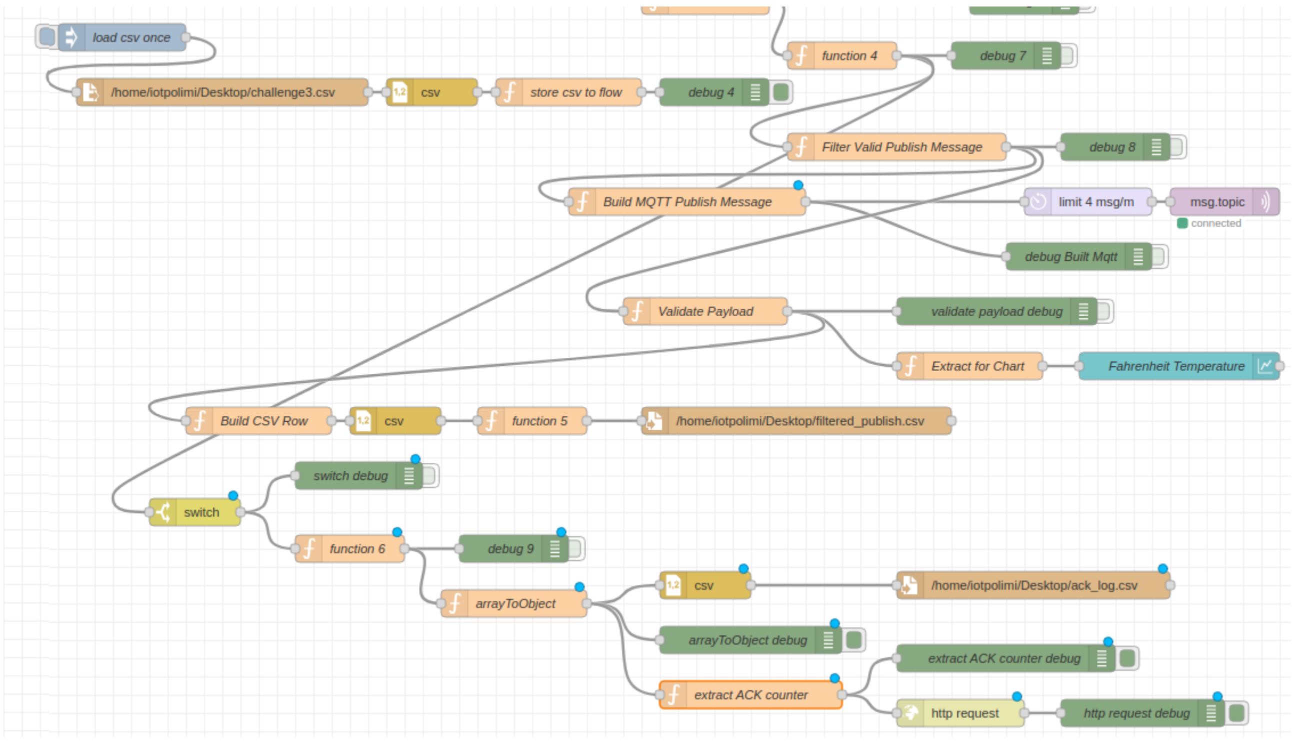
```
4/23/2025, 8:22:49 PM   node: debug 8
N value : msg.payload : Object
▾object
  No.: 7537
  Time: 2211.451877187
  Source: "10.0.2.15"
  Destination: "3.65.137.17"
  Protocol: "MQTT"
  Length: 208
  Source Port: 42827
  Destination Port: 1883
  Info: "Publish Message (id=26) [hospital/facility2/room0/deposit]"
  Payload: "{"description": "Room Temperature", "long": 57, "range": [4, 37], "lat": 87, "type": "temperature", "unit": "F"}"
4/23/2025, 8:22:49 PM   node: validate payload debug
N value : msg.payload : Object
▾object
  No.: 7537
  Time: 2211.451877187
  Source: "10.0.2.15"
  Destination: "3.65.137.17"
  Protocol: "MQTT"
  Length: 208
  Source Port: 42827
  Destination Port: 1883
  Info: "Publish Message (id=26) [hospital/facility2/room0/deposit]"
 ▸Payload: object
```

Challenge: What to do? (5)

If the message with Frame No. = N instead contains an MQTT ACK message (Publish Ack, Connect Ack, Sub/Unsub Ack), increment a global ACK counter, then save the message into a CSV file named “ack_log.csv” with the fields.

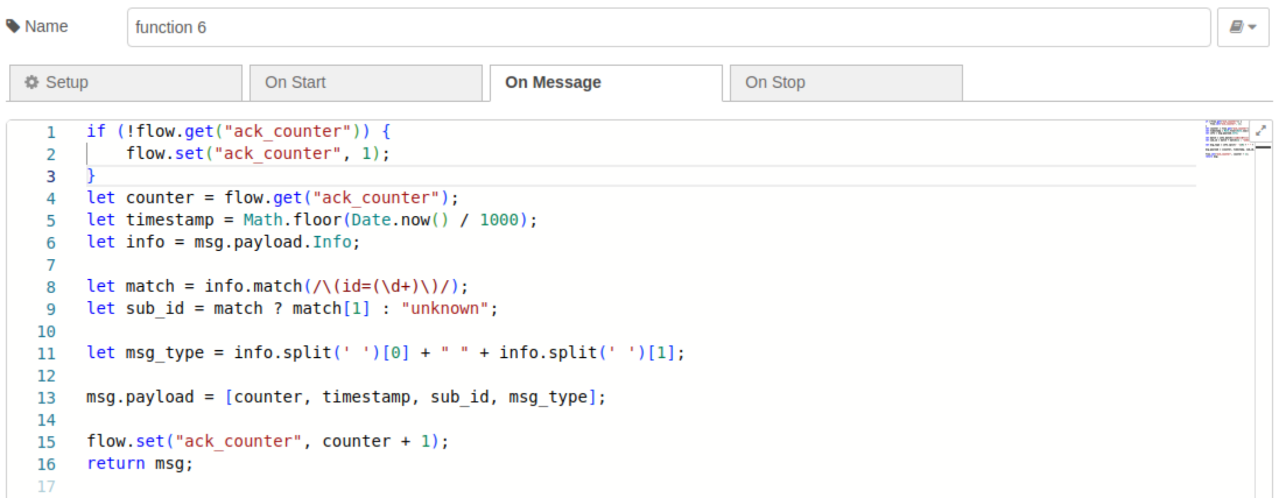
Solution:

We add these new nodes to previous flow, and This is a Thingspeak channel link : https://thingspeak.mathworks.com/channels/2925648



Switch node filters messages whose payload.Info contains the word "Ack" (e.g., "Publish Ack", "Connect Ack"). Switch debug displays messages that match the switch condition (i.e., contain "Ack" in the Info field) for monitoring and verification. Function 6 extracts structured data from the ACK message for CSV storage.

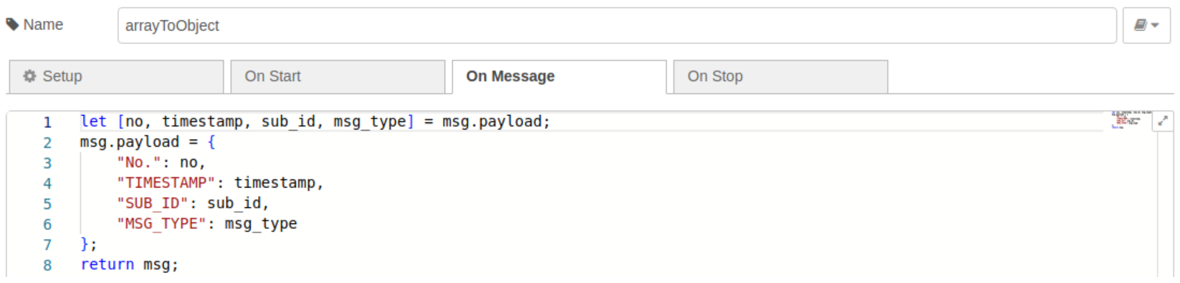
This is the code of function 6.



```
if (!flow.get("ack_counter")) {
    flow.set("ack_counter", 1);
}
let counter = flow.get("ack_counter");
let timestamp = Math.floor(Date.now() / 1000);
let info = msg.payload.Info;

let match = info.match(/\(id=(\d+)\)/);
let sub_id = match ? match[1] : "unknown";

let msg_type = info.split(' ')[0] + " " + info.split(' ')[1];

msg.payload = [counter, timestamp, sub_id, msg_type];

flow.set("ack_counter", counter + 1);
return msg;

```

Debug 9 displays the output of function 6 to confirm the correct format before saving or transforming. arrayToObject function converts the array generated by function 6 into a structured object with labeled keys.

This is the code of arrayToObject function.

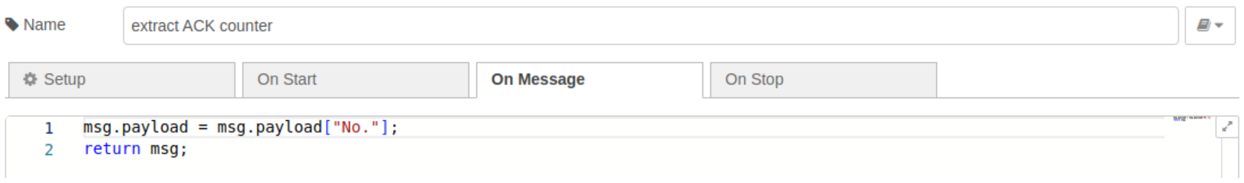


arrayToObject

```
let [no, timestamp, sub_id, msg_type] = msg.payload;
msg.payload = {
    "No.": no,
    "TIMESTAMP": timestamp,
    "SUB_ID": sub_id,
    "MSG_TYPE": msg_type
};
return msg;
```

arrayToObject debug converts the object into a CSV-formatted string with the correct header.
Csv node converts the JavaScript object from the arrayToObject function into a properly formatted CSV row. extract ACK counter extracts the No. field (i.e., the current ACK message counter) from the object and sets it as the message payload. This value is needed for updating the ThingSpeak channel.

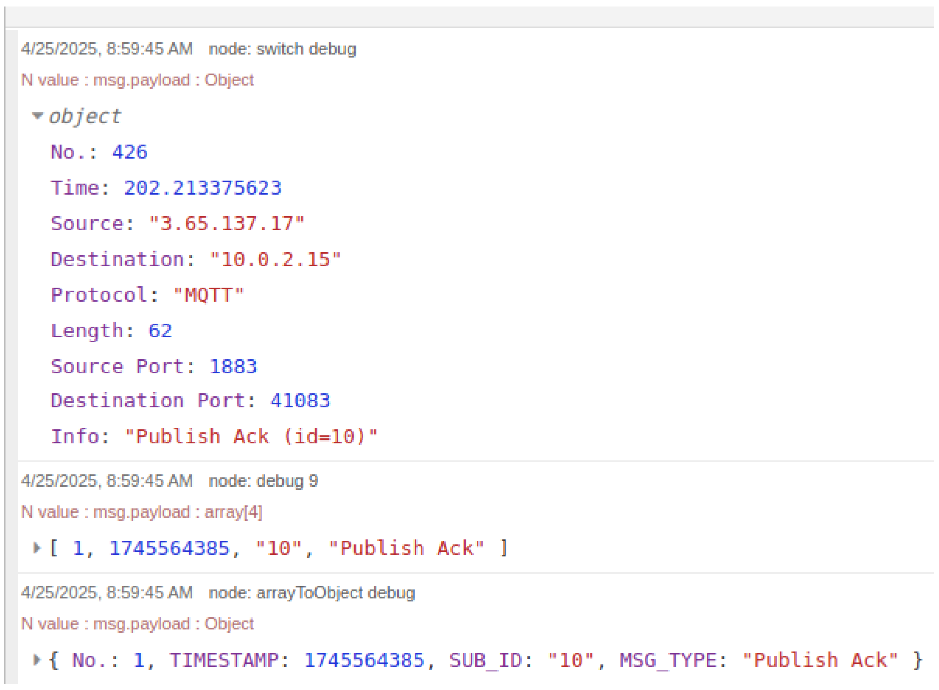
This is the code of extract ACK counter function.



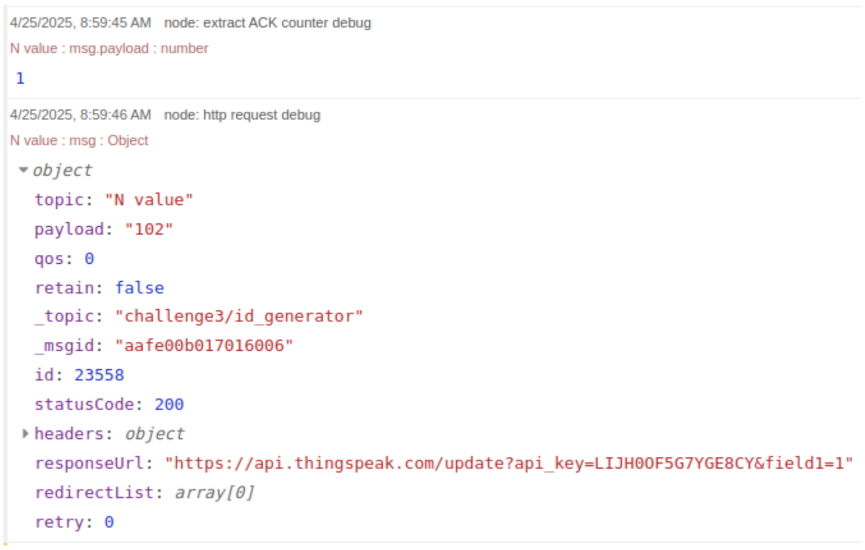
extract ACK counter

```
msg.payload = msg.payload["No."];
return msg;
```

Write file node appends the formatted CSV rows to a log file named ack_log.csv. This file records each incoming ACK message as a new row. extract ACK counter debug displays the extracted counter value that will be sent to ThingSpeak.
http request Sends a GET request to a ThingSpeak and the http request debug Shows the complete response from ThingSpeak to confirm the update was successful. There are samples of switch debug, debug 9, arrayToObject debug, extract ACK counter debug and http request debug. This is a Thingspeak channel link : https://thingspeak.mathworks.com/channels/2925648



```
4/25/2025, 8:59:45 AM   node: switch debug
N value : msg.payload : Object
▼object
  No.: 426
  Time: 202.213375623
  Source: "3.65.137.17"
  Destination: "10.0.2.15"
  Protocol: "MQTT"
  Length: 62
  Source Port: 1883
  Destination Port: 41083
  Info: "Publish Ack (id=10)"

4/25/2025, 8:59:45 AM   node: debug 9
N value : msg.payload : array[4]
▸[ 1, 1745564385, "10", "Publish Ack" ]

4/25/2025, 8:59:45 AM   node: arrayToObject debug
N value : msg.payload : Object
▸{ No.: 1, TIMESTAMP: 1745564385, SUB_ID: "10", MSG_TYPE: "Publish Ack" }
```



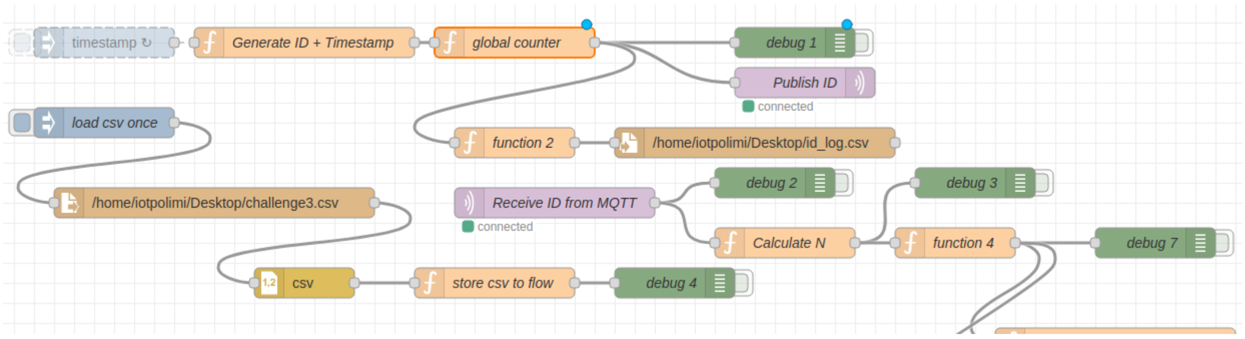
```
4/25/2025, 8:59:45 AM   node: extract ACK counter debug
N value : msg.payload : number
1
4/25/2025, 8:59:46 AM   node: http request debug
N value : msg : Object
▾object
  topic: "N value"
  payload: "102"
  qos: 0
  retain: false
  _topic: "challenge3/id_generator"
  _msgid: "aafe00b017016006"
  id: 23558
  statusCode: 200
 ▸headers: object
  responseUrl: "https://api.thingspeak.com/update?api_key=LIJH0OF5G7YGE8CY&field1=1"
  redirectList: array[0]
  retry: 0
```

Challenge: What to do? (6)

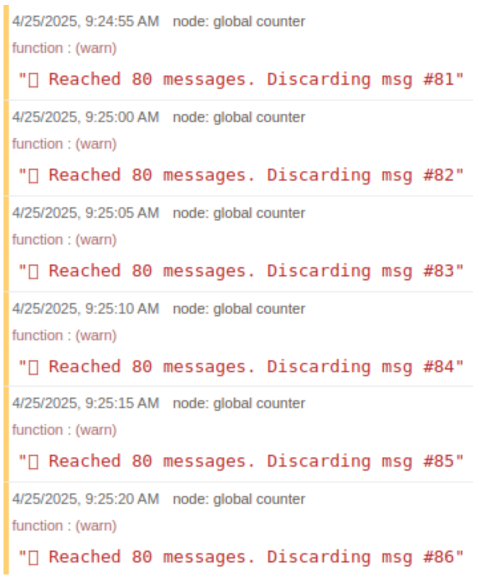
In all the other cases (frame No = N not containing an ACK or a publish) → Ignore the message! Program your flow to stop working after receiving exactly 80 id messages from the subscription: do not process more than 80 ID messages.

Solution:

We add global counter node to previous flow



Global counter function node implements a global message counter to ensure that no more than 80 messages are processed. This is the warning after 80 messages are processed.



```
4/25/2025, 9:24:55 AM   node: global counter
function : (warn)
"▯ Reached 80 messages. Discarding msg #81"
4/25/2025, 9:25:00 AM   node: global counter
function : (warn)
"▯ Reached 80 messages. Discarding msg #82"
4/25/2025, 9:25:05 AM   node: global counter
function : (warn)
"▯ Reached 80 messages. Discarding msg #83"
4/25/2025, 9:25:10 AM   node: global counter
function : (warn)
"▯ Reached 80 messages. Discarding msg #84"
4/25/2025, 9:25:15 AM   node: global counter
function : (warn)
"▯ Reached 80 messages. Discarding msg #85"
4/25/2025, 9:25:20 AM   node: global counter
function : (warn)
"▯ Reached 80 messages. Discarding msg #86"
```