

# ACTIVITY 4 – IoT

***By Stefano Agresti, Gianluca Guarro***

## PERSONAL INFO

**Name:** Stefano Agresti

**Person code:** 10685203

**Matricola:** 913079

-

**Name:** Gianluca Guarro

**Person code:** 10658177

**Matricola:** 918696

-

**Thingspeak channel:** 1066868

-

## THINGSPEAK CHANNEL

First thing to do was setting up a Thingspeak channel.

The procedure to open a new one is pretty straightforward using the GUI on their website and doesn't require any particular explanation.

Inside the channel we put two charts and two lamps. The charts show the data coming in the two different fields ("field1" and "field2"), while the two lamps light up when if the last value in their respective fields was greater than 2000.

The channel is public and reachable at:

<https://thingspeak.com/channels/1066868>.

## NODE-RED

More complicated is setting up Node-Red to send the messages.

Before moving on, we need to mention that we had to modify the csv to make it easier to parse. Using a Python script, we replaced the space separators with commas (removing further commas inside the packet messages). This step was necessary, as spaces were badly interpreted by Node-Red, and the final result can be seen in "TrafficFiltered.csv".

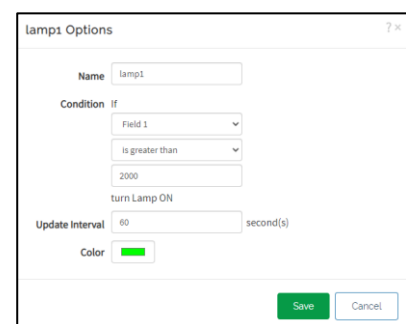


Figure 1: Setting up a lamp in Thingspeak

The final structure, shown in Figure 2, is built in this way:

- “Inject” node: by pressing the button, messages start being sent.
- “File” node: given a path to a file, it reads it, returning a string with what was read.
- “Csv” node: given a string containing a csv formatted file, returns a JSON object with the same info. We implemented to return one line at the time, in order to make the following steps easier to work on.
- “Function” node 1: filters all messages that are not of type “Publish Message”.
- “Function” node 2: filters all messages that are not about one of the four topics we’re interested in.
- “Function” node 3: converts the hexadecimal value in a string and then retrieves the sensor data.
- “Function” node 4: builds the MQTT request.
- “Rate limiter” node: up until now all messages were processed one after the other (which means, they’re all ready almost at the same time). To avoid rejections from Thingspeak, and to conform ourselves to the requirements of the homework, we need to slow down message sendings to one every minute. This is done through this node.
- “MQTT” node: connects to an MQTT broker (on deploy) and then sends messages to it. The server used is `mqtt.thingspeak.com:1883`.

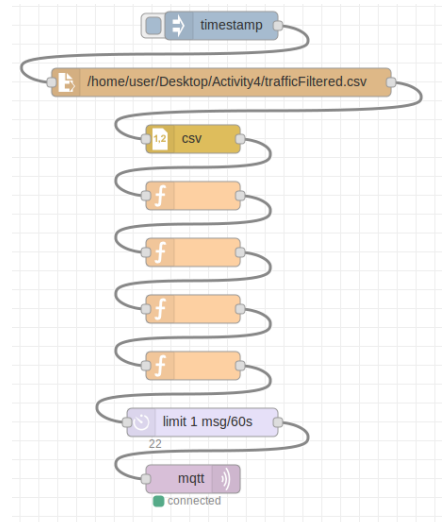


Figure 2: Final Node-Red structure

It's important to notice that there is an inherent flow in nodes 1 and 2. Using Javascript to manually parse the messages can be quite complex and prone to errors, hence, being this course about Internet of Things, we preferred to avoid a complicated mechanism and just let through messages containing the words “Publish”, “Message”, plus one of the four topics. Once this is done, the last word of the message is retrieved and returned as the message value.

However there are a number of issues that can arise this way: one message can contain more publish requests (we might therefore send the wrong value); a message might contain the words “publish” and “message”, without being a publish request; the code is generally less elegant and more difficult to maintain.

Looking at the file, we didn't notice any message that could be ambiguous, but of course this is not something a good system can rely on. To build a better and more scalable solution, our suggestion is to simply do the message selection before sending everything to Node-Red (for example using Wireshark) and, after that, using a better csv formatting so that the transformation to JSON objects is straightforward, as well as retrieving the different parts of the packet message (for example, using semicolons as column separator wouldn't create any ambiguity as they're not present inside the messages).