

Ubiquitous Computing - Lab 3: Node-RED

Vianney Hervy

1. Exercise 1

The Arduino code was not difficult to write but I had to think to organize the read and write functionnalities.

```
#include <WiFiNINA.h>

#define BUTTON_PIN 2 // see circuit in subject.pdf

int lastButtonState = LOW; // start with led off

void setup() {
    Serial.begin(9600);
    // prepare pins to use
    pinMode(LEDB, OUTPUT);
    pinMode(BUTTON_PIN, INPUT);
}

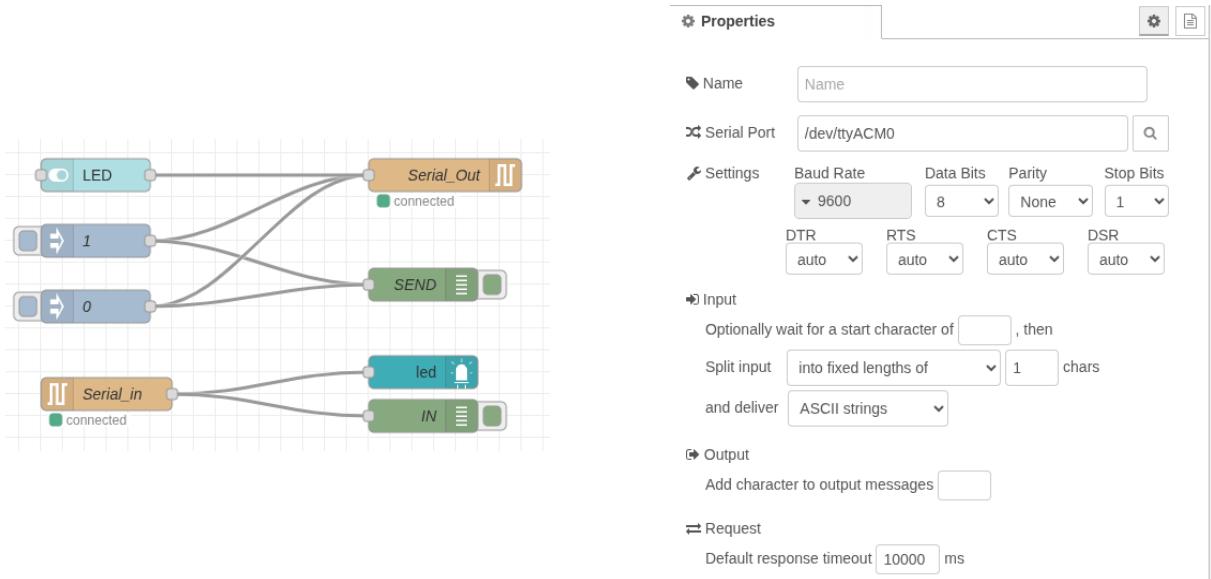
void loop() {

    // read what Node-RED sent
    if (Serial.available() > 0) {
        switch (Serial.read()) {
            case '1': // turn on light
                digitalWrite(LEDB, HIGH);
                break;
            case '0': // turn off
                digitalWrite(LEDB, LOW);
                break;
        }
    }

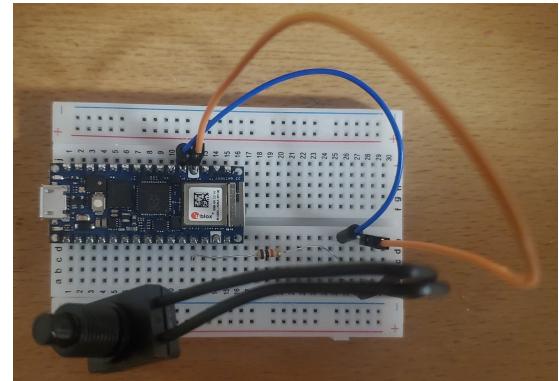
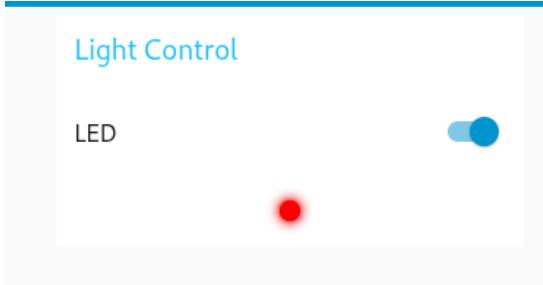
    // write to Node-RED
    int currentButtonState = digitalRead(BUTTON_PIN);
    if (currentButtonState != lastButtonState) { // only send when update
        lastButtonState = currentButtonState;
        Serial.print(currentButtonState == HIGH ? '1' : '0');
    }

    delay(100); // wait to avoid overloading Node-RED
}
```

The flow was trickier to craft. In particular, the configuration of the serial port. Indeed, the Arduino sketch was way too fast for the Node-RED flow to follow. Instead of reading the numbers one by one, it read large chunks of 000...0 and 111...1. To solve this problem, I added a delay of 100ms and a “toggle-only output” in the Arduino sketch. I also configured the serial port to split the input in chunks of 1 character.



I also configured the led node to map 1 (resp. 0) to the red (resp. gray) color, so that it looks turned on when the arduino button is pressed and turned off otherwise. The switch node was configured accordingly with string “1” when “On Payload” and “0” when “Off Payload”



Another issue I ran into was a set of GND pins that weren’t working. I eventually discovered that using my fingers worked as a temporary ground, but only after spending time testing each component to figure out which one was actually defective.

2. Exercise 2

This exercise was pretty straightforward since a similar Arduino sketch was required for the first lab session. I just changed the code for the captor to read the temperature as a float and see more subtle fluctuations on the dashboard’s chart.

```
#include <Arduino_LSM6DSOX.h>

void setup() {
  Serial.begin(9600);
  if (!IMU.begin()) {
    Serial.println("Failed to init IMU!");
    while (1)
      ;
  }
}
```

```

void loop() {
    float tempC; // use a float for better precision

    // read temperature and write to serial port
    if (IMU.temperatureAvailable()) {
        IMU.readTemperatureFloat(tempC);
        Serial.println(tempC);
    }
    delay(100);
}

```

The gauge node is configured as instructed in the subject with three different ranges.

The image shows two side-by-side configuration panels for a gauge node.

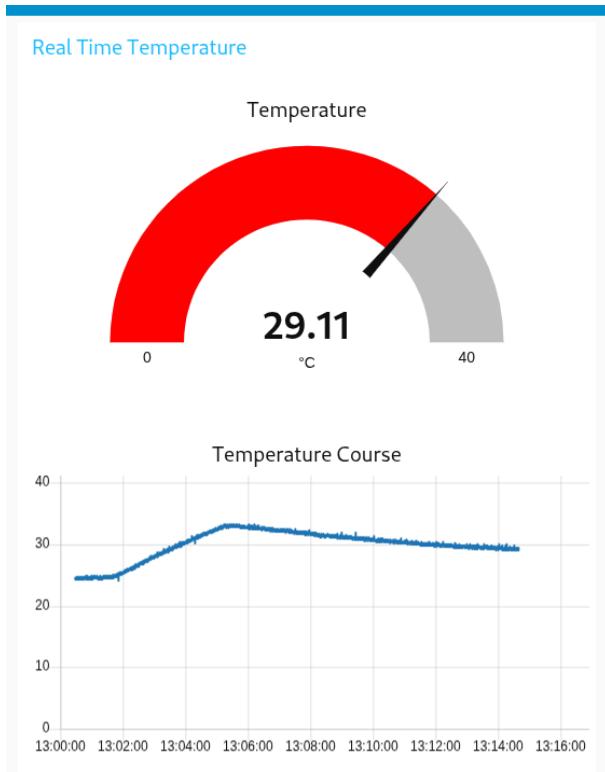
Properties Panel:

- Group:** [Temperature] Real Time Temperature
- Size:** auto
- Type:** Gauge
- Label:** Temperature
- Value format:** {{value}}
- Units:** °C
- Range:** min 0, max 40
- Colour gradient:** A horizontal bar divided into three segments: blue, green, and red.
- Sectors:** 0, ..., 15, ..., 25, ..., 40
- Fill gauge from centre:**
- Class:** Optional CSS class name(s) for widget
- Name:** Temperature

Properties (Advanced) Panel:

- Group:** [Temperature] Real Time Temperature
- Size:** auto
- Label:** Temperature Course
- Type:** Line chart enlarge points
- X-axis:** last 1 hours OR 1000 points
- X-axis Label:** HH:mm:ss as UTC
- Y-axis:** min 0, max 40
- Legend:** None Interpolate linear
- Series Colours:** A grid of six color swatches: blue, light blue, orange, green, light green, red, pink, purple, and light purple.
- Blank label:** display this text before valid data arrives
- Class:** Optional CSS class name(s) for widget
- Name:** Temperature Course

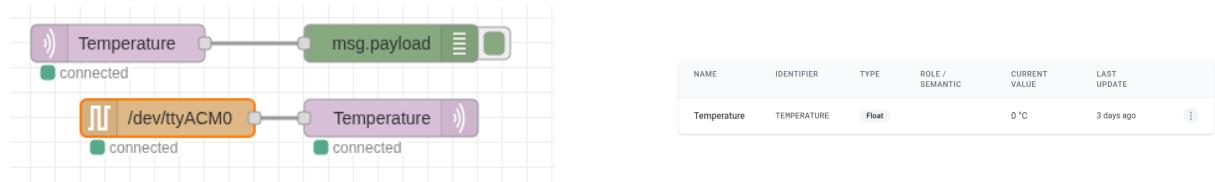
Below is the final dashboard after a few minutes of heating then cooling the Arduino board to display changes.



3. Exercise 3

This exercise's goal is to use a third element in the information chain: cloud software. The data is collected by the Arduino board's sensor as in the previous exercise (same code and serial in node), It is then sent to the MQTT broker through a dedicated node. Another dedicated node reads the data from the MQTT broker and outputs it to the debug console.

All nodes are on the same Node-RED computer, but the flow is separated in two distinct components that don't directly communicate. This means they could be running on two different instances anywhere in the world. For instance one would be in a patient's home and the other at a hospital or a data collection center.



I believe everyone had an issue with the Datacake part of this exercise. Indeed, the MQTT integration service is now a paid plan feature.

Integrations

MQTT • Deactivated

Publish and subscribe to device data via MQTT broker

This feature is only available on paid devices. Please upgrade to enable MQTT support.