

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 functionalities.

```
#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(LED_B, OUTPUT);
  pinMode(BUTTON_PIN, INPUT);
}

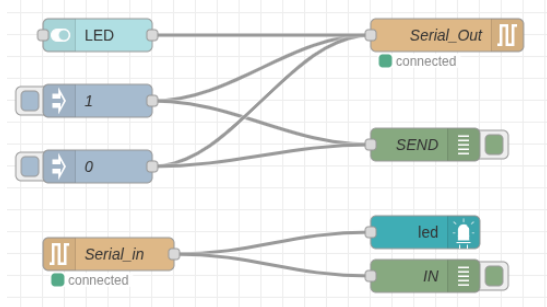
void loop() {

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



Properties

Name:

Serial Port:

Settings

Baud Rate	Data Bits	Parity	Stop Bits
9600	8	None	1

DTR: auto, RTS: auto, CTS: auto, DSR: auto

Input

Optionally wait for a start character of , then

Split input: into fixed lengths of chars

and deliver: ASCII strings

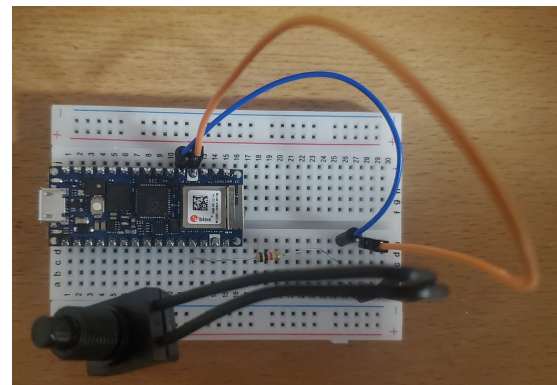
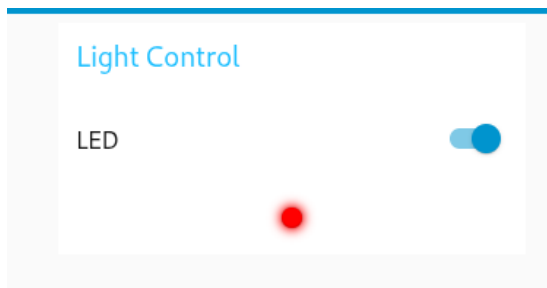
Output

Add character to output messages:

Request

Default response timeout: ms

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)
      ;
  }
}
```

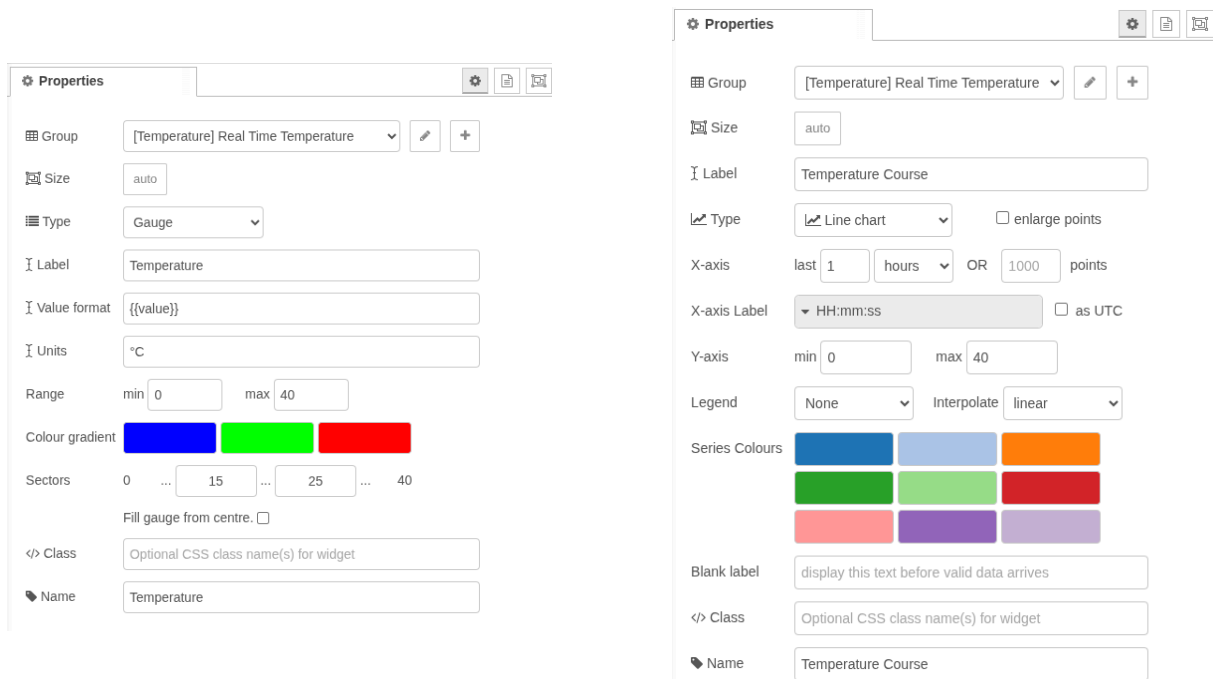
```

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

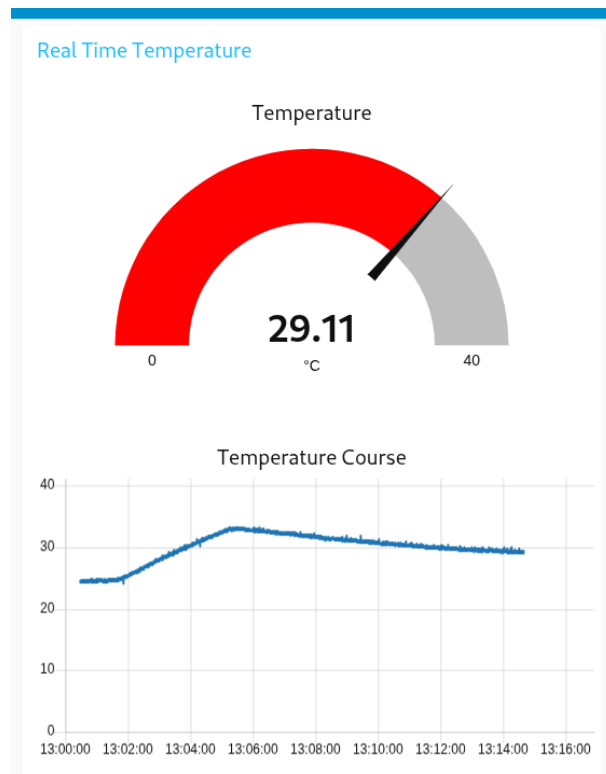
    // read temperature and write to serial port
    if (IMU.temperatureAvailable()) {
        IMU.readTemperatureFloat(tempC);
        Serial.println(tempC);
    }
    delay(100);
}

```

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



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



3. Exercise 3

TODO

```
#include <WiFiNINA.h>

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

int lastButtonState = LOW;

void setup() {
  Serial.begin(9600);
  pinMode(LED_B, OUTPUT);
  pinMode(BUTTON_PIN, INPUT);
}

void loop() {
  // read
  if (Serial.available() > 0) {
    switch (Serial.read()) {
      case '1': // turn on light
        digitalWrite(LED_B, HIGH);
        break;
      case '0': // turn off
        digitalWrite(LED_B, LOW);
        break;
    }
  }

  // write
```

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

delay(100);
}
```
