# "IoT, Interfacing with Ignition"

**California State University, Northridge**

*Department of Manufacturing Systems Engineering and Management*


**MSE 614 – Smart Manufacturing**

**Submission Date:**

**May 19, 2023**

**Professor:**

**Adrian Sita**


**By:**

**Daniel Diaz, Jack Barrientos**

# Contents

**Abstract** ... 3
**Introduction** ... 3
**Flow Diagram** ... 4
**Clients and Broker** ... 5
**Set-up for Client** ... 6
**Set-up for Inductive Ignition** ... 7
**Conclusion** ... 11
**Reflecting Troubleshooting:** ... 11
**Video Result:** ... 12
**Works Cited** ... 12
**Appendix A:** ... 14
**Appendix B:** ... 17
**Appendix C:** ... 18

## Abstract

The objective of this report is to demonstrate the interfacing application of an Internet of Things (IoT) device in conjunction with the Ignition software platform. By using the Arduino Nano 33 IoT as the primary hardware and the MQTT protocol for data transmission, the connection between the Arduino to Ignition was a success in order to enable real-time data visualization and interaction. The report details the selection of the hardware, the setup process for both hardware and Ignition, and the integration using MQTT. Additionally, a key focus of this work was the use of data binding in Ignition, enabling the dynamic manipulation of components based on incoming data from the Arduino device. While the project phase did not include the capability to send control signals from Ignition to the Arduino, it formed the basis for future exploration into such advanced IoT applications. This report will be of significant value to individuals or organizations seeking to understand and implement their own IoT projects for simple fast data collection and interpretation.

## Introduction

In this report, we aim to delve deep into the intricacies of the Internet of Things (IoT) by exploring its interface with Inductive Automation's Ignition. Ignition is a powerful industrial application platform with a feature set built around the demands of modern industrial processes, and it provides a crucial link between IoT devices and the broader data management ecosystem.

Our objective is to broaden our understanding of various client types, or 'things', that can connect via a broker and relay data to a dashboard. This foundation will lay the groundwork for us to further our exploration into big data collection and the intriguing world of digital twins.

However, before we venture into these advanced realms, it is imperative to master the basics. Hence, this report will detail the selection of devices we used and illustrate the process by which we successfully managed to interface these devices with Ignition. By enhancing our understanding of these fundamental connections, we can pave the way for more complex and expansive IoT applications in the future.

## Flow Diagram

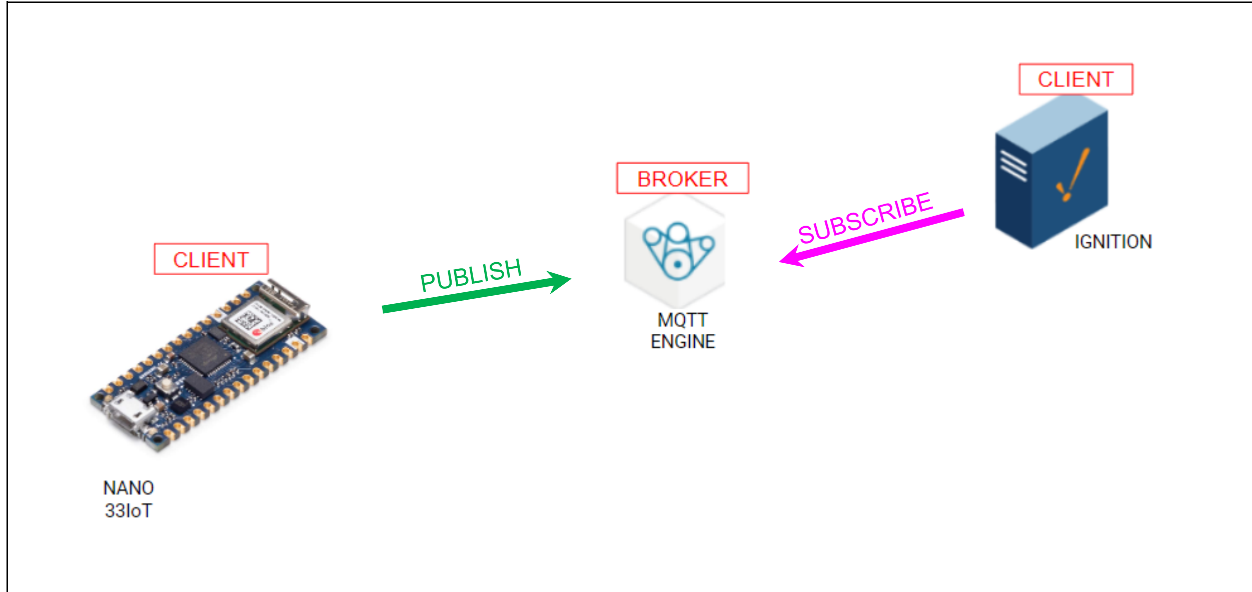


Figure 1: This flow diagram illustrates the communication process between an Arduino device and the Ignition software platform using the MQTT (Message Queuing Telemetry Transport) protocol.

The aim of this project is to transmit data through the Arduino Nano33Iot using MQTT Engine, a specific module within Ignition, thereby enabling Ignition to perform specific tasks with this data. This process involves setting up a parameter, which necessitates some coding that we will elaborate on later in this report, to define and publish topics. The MQTT broker retains the most recently published data, making it available for any client to subscribe to. Subscription, in this context, equates to data acquisition. In our particular scenario, the client Ignition is subscribing to, or obtaining the data.

Here is a logistic of the flow diagram:

1. Arduino Device:
    - Collects data from various sensors or inputs.
    - Processes and prepares the data for transmission.
    - Establishes a connection with the MQTT Broker.
    - Publishes the data to specific MQTT topics on the Broker.
2. MQTT Broker:
    - Serves as a central messaging hub for the MQTT protocol.

- Receives published data from the Arduino.
- Stores the data temporarily until it is delivered to subscribers.
- Manages subscriptions to different MQTT topics.

3. Ignition Software Platform:
    - Establishes a connection with the MQTT Broker.
    - Subscribes to specific MQTT topics on the Broker.
    - Receives data from the Broker when published by the Arduino.
    - Processes and utilizes the received data within Ignition.

## Clients and Broker

Selecting the appropriate hardware can be a challenging task due to the myriad of options available. The primary factor to consider is whether the hardware possesses WiFi capability, as this is often indicative of its suitability for our needs. We required a simple yet effective device capable of handling three buttons, three LED lights, and a time tracker for generating numbers. Our initial choice was the Arduino Uno, due to its immediate availability. However, the need for an additional module hardware, WiFi connectivity, presented a complication. After further investigation, we settled on the Arduino Nano 33 Iot. See figure 2, a Microcomputer that was chosen. A detailed comparison of different boards can be found in Appendix A.

As for identifying a suitable broker for our project presented its own set of challenges. Initially, we attempted to establish a local broker on a desktop using Mosquitto. However, this solution fell short of our needs because the broker would stop functioning when the desktop was powered off. It's worth mentioning that running a dedicated server using Raspberry Pi is a possible solution. While we won't get into the specifics of that option in this report, a wealth of information on setting up a Raspberry Pi as a dedicated MQTT broker server is readily available online. Ultimately, due to our need for a continually available server and without needing a dedicated server, we opted for the MQTT Engine Module provided by Inductive Automation. We have included a thorough comparison of various MQTT brokers in Appendix B for those seeking additional insight into potential options.

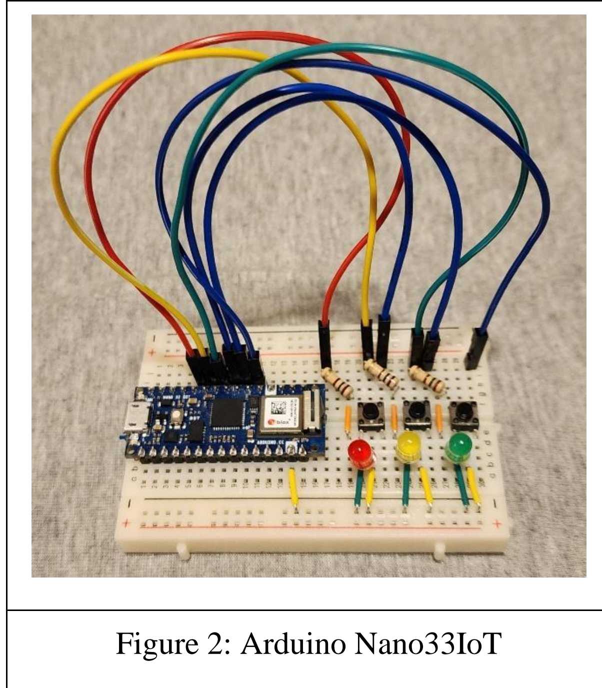

Figure 2: Arduino Nano33IoT

## Set-up for Client

As we've previously discussed, we chose to use the Arduino Nano 33 IoT for our project. While we won't delve into the granular details of the hardware and coding, we'll outline the used pins, the purpose of our code, and its key elements. The complete code can be found in Appendix C.

First, it's crucial to note that the libraries for WiFiNINA and ArduinoMqttClient need to be installed, as we've utilized their syntax in our code (see Figure 3, Lines 1 & 2). Lines 4 & 5 establish the WiFi connectivity. Here, 'SSID' represents your WiFi name, and 'password' stands for your corresponding WiFi password. Please remember to retain the quotation marks as they denote the text as a string. Line 9 identifies the IP address of the broker (refer to Appendix B for free server options) and Line 10 specifies the MQTT port.

Our choice of data or 'Topics' to be sent to the broker can be seen in Lines 13 to 17 of our code. These topics include status data for when the Green, Red, or Yellow buttons are clicked, and some additional variables like 'room 1' and 'room 2', which are included for practice. The topics are sent as random numbers, for simplicity's sake, within a specified range. The naming scheme 'Arduino/(topic)' is designed to maintain organization: the prefix 'Arduino/' can be considered a folder on the dashboard, enabling any client to subscribe to all 'Arduino/' topics by

subscribing to 'Arduino/#'. The '#' symbol acts as a wildcard. We will elaborate more on this in the Inductive setup section.

For the hardware setup, please refer to Figure 2. We've assigned Pins 2, 3, and 4 on the Arduino Nano 33 IoT to use buttons to control the Green, Yellow, and Red LEDs respectively. These LEDs are connected to Pins 5, 6, and 7. To regulate the current and protect the LEDs, we've incorporated a 100-ohm resistor in series with each LED, connected to the positive side (anode) of the LEDs.

Nano33iot_MQTT_Ignition_03.ino

```
1  #include <WiFiNINA.h>
2  #include <ArduinoMqttClient.h>
3
4  // network credentials
5  const char* ssid = "XXXXXXXXX";
6  const char* password = "XXXXXXXXX";
7
8  // MQTT server IP address
9  const char* mqttServer = "XXXXXXXXX";
10 int mqttPort = 1883;
11
12 // Set your desired MQTT topics
13 const char* mqttRandomTopic1 = "arduino/room1";
14 const char* mqttRandomTopic2 = "arduino/room2";
15 const char* mqttButtonTopicG = "arduino/buttonGreen";
16 const char* mqttButtonTopicR = "arduino/buttonRed";
17 const char* mqttButtonTopicY = "arduino/buttonYellow";
```

Figure 3: Displaying 17 lines of code to show the important ones. Library for Wifinina can be found, https://www.arduino.cc/reference/en/libraries/wifinina/ and ArduinoMqttClent https://www.arduino.cc/reference/en/libraries/arduinomqttclient/

## Set-up for Inductive Ignition

In our project, the dashboard we created simulates real-world events, hence facilitating real-time data collection. To achieve this, we used specific perspective components and bound data to them. This setup means components respond only if certain conditions are met, as defined by the received data.

Let's look at Figure 4.1 as an example: here, we dragged and dropped the tag. In Figure 4.2, we added a label, leaving its name blank, which resulted in a blank square object (an improvised solution due to the absence of a simple circle). Figure 4.3 showcases styles, essentially variables containing custom properties. The advantage here is the ability to link these styles to any components. Instead of adjusting

each component individually, you can edit all linked components simultaneously via the style variable. Let's delve deeper into 4.1, 4.2, and 4.3.

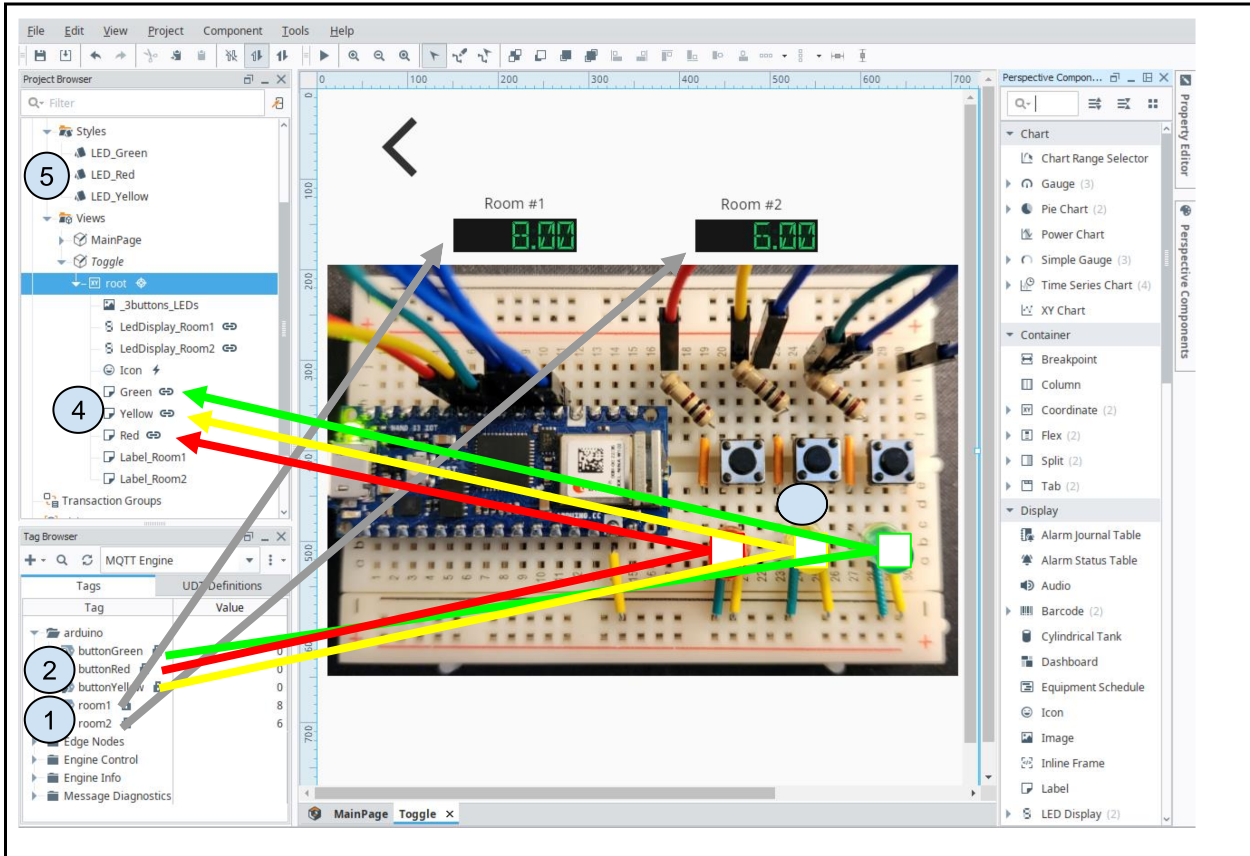


Figure 4: Visual Display, essentially a simulation depending on data received.

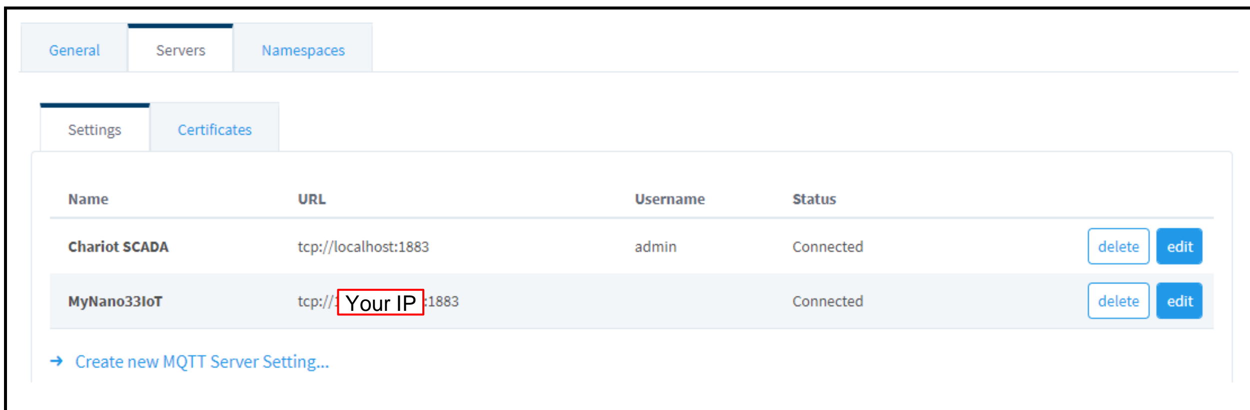


Figure 5: Servers, the URL should have your broker address, if not then create a new one and add it. Documentation, see https://docs.chariot.io/display/CLD80/ME%3A+Configuration

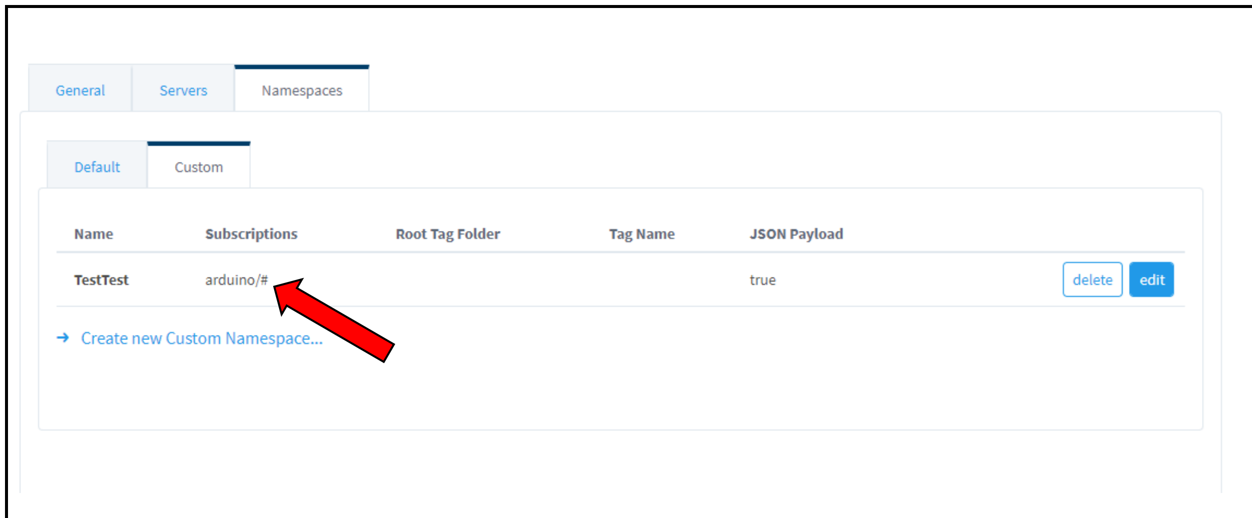


Figure 6: We are subscribing to anything that has arduino/, see figure 4 and notice the folder "arduino". So, when subscribing, the first word will essentially be a folder, with that said, plan accordingly.

In Figures 4.1 & 4.2, it's crucial that you set your tags' dropdown to MQTT Engine, ensuring that the subscribed tags can be identified. To subscribe to topics, the Broker must first be set up. You can do this from the Ignition homepage > Config > MQTT Engine > Setting. Make sure the Broker's IP address is connected, as shown in Figure 5. To subscribe to the desired topic, as seen in Figure 6, you'll need to create a new namespace. Once you've subscribed, visible tags should appear. With these data tags, you can initiate actions – in our case, we dragged and dropped tags onto the canvas, creating LED displays. This allowed us to observe the numbers changing every three seconds.

Moving to Figure 4.3, we created labels, as seen in Figure 4.4, which shows that labels were established and the link icon indicates that the data is bound to a specific component. We exploited this binding to simulate red, yellow, and green lights reflecting the state of physical buttons. We then animated this simulation by creating styles and binding them to style classes in the perspective property.

To create a style, refer to Figure 7.1, and name it as shown in Figure 7.2. You can change the background color as seen in Figure 7.3. To create a spectrum of colors, select 7.3.1 for one call and select 7.3.2 to animate the other call. Importantly, Figure 7.4 allows for the creation of transitions from selected properties at 0% to 100% as shown 7.3.1 & 7.3.2, so be sure to enable it. Once the style is created, bind it to a tag by selecting a component and clicking on the link symbol within the Property Editor (Figure 8.1). In Figure 8.2, select the tag you want to bind, remember to select its value because we want the component to read this tag's value.

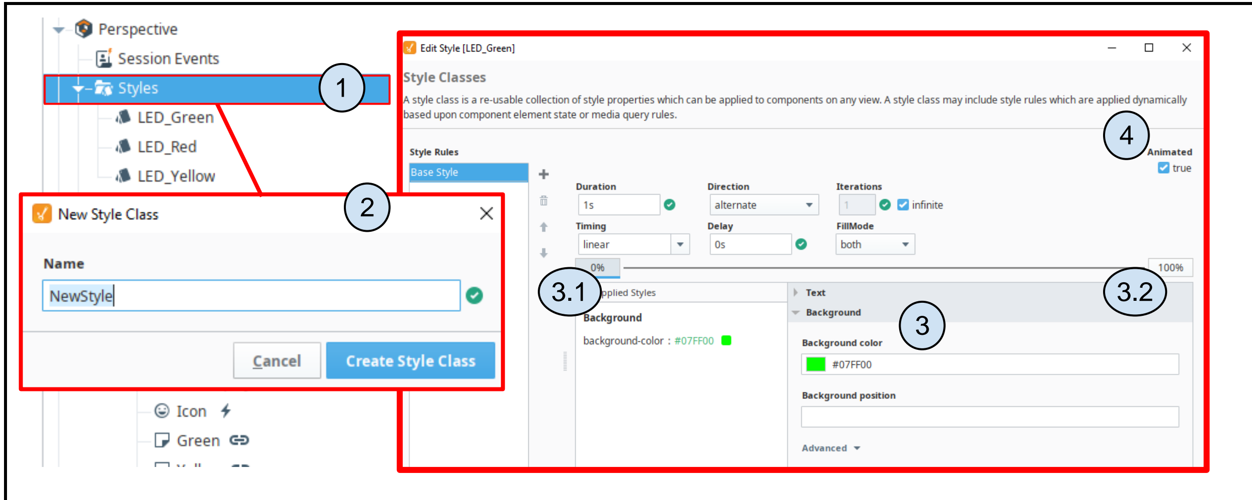


Figure 7: Process of creating a style.

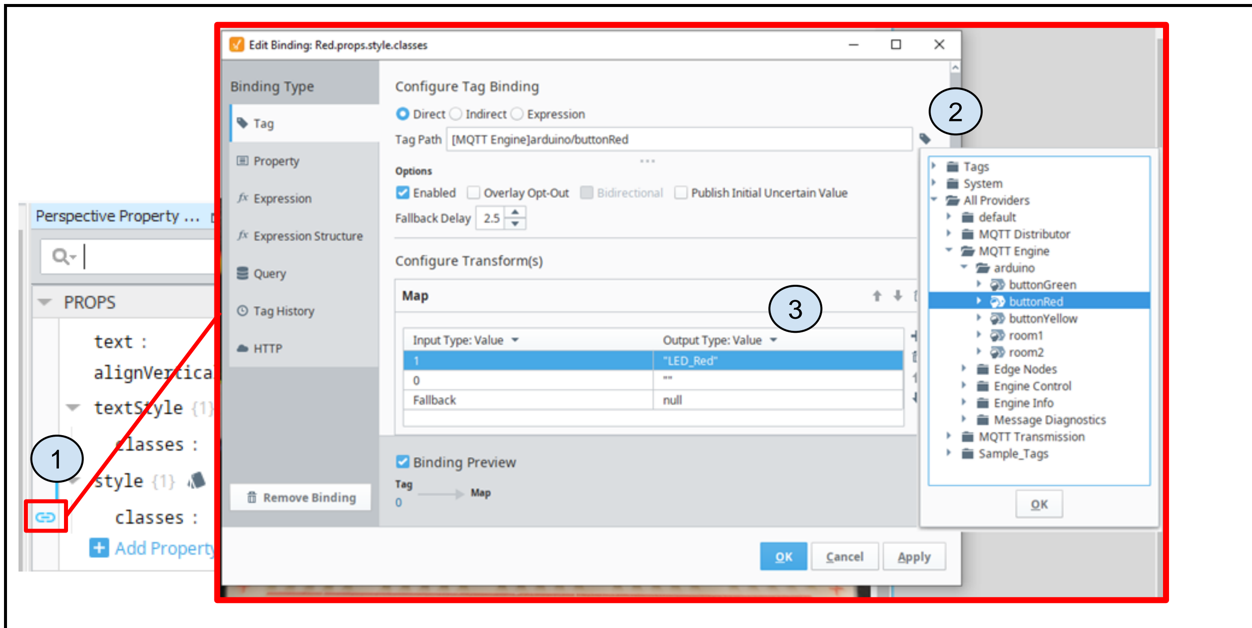


Figure 8: Process of binding style and tag

Finally, for Figure 8.3, we also bind the style created in Figure 7. Initialize the binding by selecting transform and creating two lines (not shown here as it has already been done). The component is then programmed to respond when the input reads '1', activating the style 'LED Green' properties to match accordingly. When the input reads '0', it reverts to the default, which is blank.

This demonstration shows the power of binding, specifically how we utilized the inputs of the button in its on/off states. The potential applications are as limitless as your imagination.

## Conclusion

In conclusion, the project provided a foundational understanding of the communication between the Arduino Nano 33 IoT device and Ignition using MQTT. Our successful implementation showcased the seamless integration of hardware, the MQTT protocol, and Ignition's powerful features for visualizing and managing real-time data. We were able to subscribe to data topics from the Arduino device, binding this incoming data to manipulate components on the Ignition platform dynamically. Although this phase of the project did not encompass two-way communication, enabling control signals to be sent from Ignition to the Arduino, our investigation lays the groundwork for this next step. We now have a clear path towards expanding our knowledge and exploring more complex IoT applications, including two-way control and the creation of digital twins.

## Reflecting Troubleshooting:

Previously, it was discussed briefly on how to get the MQTT broker to work. However, the troubles that weren't discussed was the process of setting up a dedicated MQTT broker server and validating topic publication through a third-party MQTT client. The approach varied depending on the board used, which included esp-8266, esp-32 CAM, Arduino MKR-1000, and Arduino Nano. In short, the key to these variations was understanding the specific libraries each board utilized.

Our initial goal was to establish a two-way read/write communication system, but unfortunately, we were not successful. Consequently, we resorted to implementing a one-way system from Arduino to Ignition. Initially, we aimed to control a 6 Degree of Freedom (DoF) Robot. However, due to time constraints and the difficulties faced in establishing a two-way system, we decided to realign our objectives. We focused on understanding and mastering the basics first, thus our project pivoted to facilitate one-way communication from Arduino to Ignition. This experience opened the possibility of future work, particularly in resolving the two-way communication challenge and progressing towards controlling complex systems such as the 6 DoF Robot.

In terms of coding, we encountered some problems. Specifically, the button didn't respond as expected. We had a setup where a random number generator would send data at fixed intervals, but integrating this with the button proved problematic. Our original code required a

button press before generating random numbers, which was not ideal. To rectify this, we introduced a time condition. This modification helped keep track of time and allowed the code to complete loops independently of the button presses, improving the response time and overall performance of the system.

## Video Result:

Click to See video
direct link, viewable for anyone with link:
https://drive.google.com/file/d/1szn3ANM0jmxxK6ekGCEb2YeEcTonzLB7/view?usp=share_link

## Works Cited


"ArduinoMqttClient." *Arduino*, https://reference.arduino.cc/reference/en/libraries/arduinomqttclient/. Accessed May 2023.

"Arduino Tutorial 28: Using a Pushbutton as a Toggle Switch." *YouTube*, Paul McWhorter, 17 September 2019, https://www.youtube.com/watch?v=aMato4olzi8. Accessed May 2023.

Davenport, Nathan. "MQTT Engine - MQTT Modules for Ignition 8.x - Confluence." *Cirrus Link Documentation*, 15 September 2021, https://docs.chariot.io/display/CLD80/MQTT+Engine. Accessed 19 May 2023.

"Inductive University." *" - Wiktionary*, https://www.inductiveuniversity.com/courses/ignition/perspective-components-and-bindings/8.1. Accessed 19 May 2023.

Richetta, Andrea. "Nano 33 IoT." *Arduino*, https://docs.arduino.cc/hardware/nano-33-iot. Accessed May 2023.

“WiFiNINA.” *Arduino*, https://www.arduino.cc/reference/en/libraries/wifinina/.
Accessed May 2023.

# Appendix A:

| Board | Advantages | Disadvantages | Application | # of ADC pins |
|---|---|---|---|---|
| Arduino Nano 33 IoT | • Compact form factor, similar to the original Arduino Nano.<br>• Easy to use with Arduino IDE and extensive community support.<br>• Built-in WiFi and Bluetooth for IoT applications.<br>• 3-axis accelerometer and 3-axis gyroscope.<br>• 3.3V I/O pins.<br>• Lower power consumption, suitable for battery-powered projects. | • Compact form factor, similar to the original Arduino Nano.<br>• Easy to use with Arduino IDE and extensive community support.<br>• Built-in WiFi and Bluetooth for IoT applications.<br>• 3-axis accelerometer and 3-axis gyroscope.<br>• 3.3V I/O pins.<br>• Lower power consumption, suitable for battery-powered projects. | • Compact form factor, similar to the original Arduino Nano.<br>• Easy to use with Arduino IDE and extensive community support.<br>• Built-in WiFi and Bluetooth for IoT applications.<br>• 3-axis accelerometer and 3-axis gyroscope.<br>• 3.3V I/O pins.<br>• Lower power consumption, suitable for battery-powered projects.<br>• | • 8 |
| ESP8266 NodeMCU (V1, V2, V3) | • Low cost<br>• Easy to use with Arduino IDE<br>• Built-in WiFi | • Limited GPIOs<br>• Less processing power<br>• Limited RAM and Flash memory<br>• No Bluetooth | • IoT devices<br>• Home automation<br>• Remote sensors<br>• Web servers | 1 |
| ESP32 DevKitC (V1, V2, V3) | • More GPIOs than ESP8266<br>• Dual-core processor<br>• Built-in WiFi and Bluetooth<br>• More RAM and Flash memory<br>• Easy to use with Arduino IDE | • Slightly higher cost<br>• Higher power consumption | • IoT devices<br>• Home automation<br>• Robotics<br>• Wearable devices<br>• Drones | 18 |
| ESP32 WROVER | • Dual-core processor<br>• Built-in WiFi and Bluetooth | • Higher cost than ESP32 DevKitC | • IoT devices with high data processing | 18 |

| | | | | |
|---|---|---|---|---|
| | • More RAM and Flash memory<br>• Easy to use with Arduino IDE<br>• Camera interface, SD card support<br>• Additional PSRAM | | • Advanced home automation<br>• Machine learning projects<br>• Image processing | |
| ESP32-CAM | • Compact form factor<br>• Built-in camera module<br>• Built-in WiFi<br>• Low cost<br>• SD card support | • Limited GPIOs<br>• No USB interface<br>• Requires external programmer | • Surveillance cameras<br>• Face recognition<br>• Object tracking<br>• Robotics<br>• Drones | 1 (Not exposed on the board, accessible through the pin header) |
| Arduino Uno | • Easy to use with Arduino IDE<br>• Large community and libraries<br>• Robust 5V I/O pins | • Limited GPIOs<br>• No built-in WiFi or Bluetooth<br>• Limited RAM and Flash memory<br>• Less processing power | • Education<br>• Hobbyist projects<br>• Prototyping<br>• Simple robotics | 6 |
| Arduino MKR1000 | • Built-in WiFi<br>• Easy to use with Arduino IDE<br>• More RAM and Flash memory than Uno<br>• Low power consumption<br>• Large community and libraries | • Higher cost than Arduino Uno<br>• Limited GPIOs<br>• 3.3V I/O pins | • IoT devices<br>• Home automation<br>• Remote sensors<br>• Battery-powered devices<br>• Wireless communication | 7 |

· References:


o ESP8266 NodeMCU:

Official GitHub Repository: https://github.com/nodemcu/nodemcu-devkit

ESP8266 Datasheet: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

o ESP32 DevKitC:

Official Espressif Documentation: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html

ESP32 Datasheet:
https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

- ESP32 WROVER:

  Official Espressif Documentation: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/modules-and-boards.html#esp32-wrover-module

  ESP32 WROVER Datasheet:
  https://www.espressif.com/sites/default/files/documentation/esp32-wrover_datasheet_en.pdf

- ESP32-CAM:

  ESP32-CAM AI-Thinker Datasheet: https://www.ai-thinker.com/uploads/file/191112193120-0/ESP32-CAMProductSpecification.pdf

- Arduino Uno:

  Official Arduino Website: https://store.arduino.cc/usa/arduino-uno-rev3

  ATmega328P Datasheet:
  http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf

- Arduino MKR1000:

  Official Arduino Website: https://store.arduino.cc/usa/arduino-mkr1000-wifi

  SAMD21 Datasheet:
  http://ww1.microchip.com/downloads/en/DeviceDoc/SAM_D21_DA1_Family_DataSheet_DS40001882F.pdf

  ATWINC1500 Datasheet:
  http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42420-WINC1500-Low-Power-2.4GHz-IEEE-802.11-b-g-n-IoT-Network-Controller_Datasheet.pdf

- Arduino Nano 33 IoT

  Main Page: https://docs.arduino.cc/hardware/nano-33-iot?queryID=9d81d5fbd484e4c5018dc8ff78d60dec&_gl=1*lrjgnd*_ga*MjM1MDg3NjAyLjE2ODA2NjA1NjE.*_ga_NEXN8H46L5*MTY4NDM3NzAzNy4xNS4xLjE2ODQzNzcxNjAuMC4wLjA.

  Technical Spec:
  https://docs.arduino.cc/static/97dd3221a167cace69dcd032870e0d57/ABX00027-datasheet.pdf

# Appendix B:

| MQTT Broker | Advantages | Disadvantages | Applications |
|---|---|---|---|
| Mosquitto | • Open-source and free<br>• Lightweight<br>• Easy to set up and configure<br>• Supports MQTT v3.1 and v3.1.1 (with limited support for MQTT v5) | • Limited features compared to other brokers<br>• No built-in web interface for management | • Hobbyist projects<br>• Small-scale deployments<br>• Resource-constrained devices (e.g., Raspberry Pi) |
| HiveMQ (Community Edition) | • Hobbyist projects<br>• Small-scale deployments<br>• Resource-constrained devices (e.g., Raspberry Pi) | • Requires more resources than Mosquitto<br>• No built-in web interface for management | • Small to medium-scale deployments<br>• IoT applications requiring advanced features and scalability |
| HiveMQ (Enterprise Edition) | • Advanced security features<br>• Web-based user interface for monitoring and management<br>• Plugin system for extending functionality<br>• Commercial support available | • Commercial product with associated costs<br>• Requires more resources than Mosquitto | • Large-scale or enterprise deployments<br>• IoT applications requiring high reliability, scalability, and security |
| AWS IoT Core | • Managed cloud service<br>• High scalability and reliability<br>• Integration with other AWS services<br>• Supports MQTT v3.1.1, as well as WebSocket connections<br>• Advanced security features | • Pay-as-you-go pricing model<br>• Requires an internet connection for clients to connect<br>• Learning curve to set up and manage | • IoT applications in cloud environments<br>• Integrations with AWS services<br>• Large-scale or enterprise deployments |

Trying to find a public server where you don't need to set up an account? Here are a few public brokers, thanks to Professor Adrian Sita who mentioned a couple. Disclaimer: these public brokers are intended for development and testing. For production use, it's recommended to set up your own MQTT broker or use a managed service from a cloud provider:

- Mosquitto Test Server
  - Host: test.mosquitto.org
  - Ports: 1883 (unsecured), 8883 (TLS)
- HiveMQ Public Broker
  - Host: broker.hivemq.com
  - Port: 1883
- EMQ X Public Broker

- Host: broker.emqx.io
- Port: 1883

▪ Eclipse IoT MQTT Sandbox
  - Host: iot.eclipse.org
  - Ports: 1883 (unsecured), 8883 (TLS)

# Appendix C:

```
#include <WiFiNINA.h>
#include <ArduinoMqttClient.h>

// network credentials
const char* ssid = "XXXXXXXXX";
const char* password = "XXXXXXXXX";

// MQTT server IP address
const char* mqttServer = "XXXXXXXXX";
int mqttPort = 1883;

// Set your desired MQTT topics
const char* mqttRandomTopic1 = "arduino/room1";
const char* mqttRandomTopic2 = "arduino/room2";
const char* mqttButtonTopicG = "arduino/buttonGreen";
const char* mqttButtonTopicR = "arduino/buttonRed";
const char* mqttButtonTopicY = "arduino/buttonYellow";

// Set up the MQTT client
WiFiClient net;
MqttClient mqttClient(net);

// Set up the button and LED pins variables for Green
const int buttonPinG = 2; // Button connected to pin 2
const int ledPinG = 5;
int buttonStateG = 0;
int lastButtonStateG = 1;
int ledStateG = 0;

// Set up the button and LED pins variables for Yellow
const int buttonPinY = 3; // Button connected to pin 3
const int ledPinY = 6;
```

```
int buttonStateY = 0;
int lastButtonStateY = 1;
int ledStateY = 0;

// Set up the button and LED pins variables for Red
const int buttonPinR = 4; // Button connected to pin 4
const int ledPinR = 7;
int buttonStateR = 0;
int lastButtonStateR = 1;
int ledStateR = 0;

// delay time for LED, this allow button to "bounce" and ignore the noise when pressing, see line 98
int dt = 50;

void setup() {
  Serial.begin(9600);
  while (!Serial);

  connectToWiFi();
  connectToMqttServer();
//Green
  pinMode(buttonPinG, INPUT_PULLUP); // Set the button pin as input with internal pull-up resistor
  pinMode(ledPinG, OUTPUT); // Set the Green LED pin as output
//Yellow
  pinMode(buttonPinY, INPUT_PULLUP); // Set the button pin as input with internal pull-up resistor
  pinMode(ledPinY, OUTPUT); // Set the Yellow LED pin as output
//Red
  pinMode(buttonPinR, INPUT_PULLUP); // Set the button pin as input with internal pull-up resistor
  pinMode(ledPinR, OUTPUT); // Set the Red LED pin as output
}

// Initialize a variable to store the last time random numbers were generated
unsigned long lastRandomNumberGenerationTime = 0;
// Set the desired random number generation interval (3 seconds)
unsigned long randomNumberGenerationInterval = 3000;

void loop() {
  mqttClient.poll();
```

```
  if (!mqttClient.connected()) {
    connectToMqttServer();
  }
//GreeN
  // Read the state of the button for Green
  buttonStateG = digitalRead(buttonPinG);

  // If the button state has changed and it's HIGH (released), toggle the Green LED
  if (lastButtonStateG == 0 && buttonStateG == 1) {
    if (ledStateG == 0) {
      digitalWrite(ledPinG, HIGH);
      ledStateG = 1;
    }
    else {
      digitalWrite(ledPinG, LOW);
      ledStateG = 0;
    }

    mqttClient.beginMessage(mqttButtonTopicG);
    mqttClient.print(ledStateG);
    mqttClient.endMessage();

    Serial.println("Button pressed, Green LED state toggled!");
  }

  lastButtonStateG = buttonStateG;
  delay(dt);

//Yellow
    // Read the state of the button for Yellow
  buttonStateY = digitalRead(buttonPinY);

  // If the button state has changed and it's HIGH (released), toggle the Yellow LED
  if (lastButtonStateY == 0 && buttonStateY == 1) {
    if (ledStateY == 0) {
      digitalWrite(ledPinY, HIGH);
      ledStateY = 1;
    }
    else {
```

```
    digitalWrite(ledPinY, LOW);
    ledStateY = 0;
   }

  mqttClient.beginMessage(mqttButtonTopicY);
  mqttClient.print(ledStateY);
  mqttClient.endMessage();

  Serial.println("Button pressed, Yellow LED state toggled!");
 }

 lastButtonStateY = buttonStateY;
 delay(dt);
//Red
  // Read the state of the button for Red
 buttonStateR = digitalRead(buttonPinR);

 // If the button state has changed and it's HIGH (released), toggle the Red LED
 if (lastButtonStateR == 0 && buttonStateR == 1) {
  if (ledStateR == 0) {
    digitalWrite(ledPinR, HIGH);
    ledStateR = 1;
   }
  else {
    digitalWrite(ledPinR, LOW);
    ledStateR = 0;
   }

  mqttClient.beginMessage(mqttButtonTopicR);
  mqttClient.print(ledStateR);
  mqttClient.endMessage();

  Serial.println("Button pressed, Red LED state toggled!");
 }

 lastButtonStateR = buttonStateR;
 delay(dt);

 // Check if it's time to generate random numbers
```

```
  // This is important so when we pressed the button, we don't have to wait for numbers to be generated first
before LED light to turn on
  unsigned long currentTime = millis();
  if (currentTime - lastRandomNumberGenerationTime >= randomNumberGenerationInterval) {
    // Generate random numbers and publish to the MQTT topics
    int randomNumber1 = random(2, 9);
    mqttClient.beginMessage(mqttRandomTopic1);
    mqttClient.print(randomNumber1);
    mqttClient.endMessage();

    // Print the published random number 1 to the Serial Monitor
    // This is to debug to make sure we are publishing
    Serial.print("Published room number 1: ");
    Serial.println(randomNumber1);

    int randomNumber2 = random(6, 21);
    mqttClient.beginMessage(mqttRandomTopic2);
    mqttClient.print(randomNumber2);
    mqttClient.endMessage();

    // Print the published random number 2 to the Serial Monitor
    // This is to debug to make sure we are publishing
    Serial.print("Published room number 2: ");
    Serial.println(randomNumber2);

    // Update the last random number generation time
    lastRandomNumberGenerationTime = currentTime;
  }
}


// Connect to WiFi
void connectToWiFi() {
  Serial.print("Attempting to connect to WiFi...");
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
```

```
  }
  // This is to debug to ensure connection to WiFi
  Serial.println("Connected to WiFi!");
}

// Connect to MQTT Broker
void connectToMqttServer() {
  Serial.print("Attempting to connect to MQTT server...");

  mqttClient.setId("ArduinoNano33IoT");

  while (!mqttClient.connect(mqttServer, mqttPort)) {
    delay(500);
    Serial.print(".");
  }
  // This is to debug to ensure connection to MQTT Broker
  Serial.println("Connected to MQTT server!");
}
```