**CONTROL LED USING MQTT WITH SINRIC PRO**

**ABSTRACT:**

This project presents the seamless integration of an IoT device utilizing the ESP32 microcontroller with both MQTT (Message Queuing Telemetry Transport) protocol and SinricPro, a comprehensive cloud service tailored for IoT devices. The primary objective is to enable remote control and synchronization of an LED connected to the ESP32, thereby facilitating versatile usage scenarios ranging from home automation to industrial applications.

The integration is achieved through meticulous software development and configuration. The MQTT client facilitates bidirectional communication between the IoT device and a centralized broker, enabling users to remotely control the LED state via MQTT messages. Concurrently, the SinricPro library orchestrates cloud-based control, empowering users with voice commands and integration possibilities with leading smart home platforms such as Amazon Alexa and Google Home.

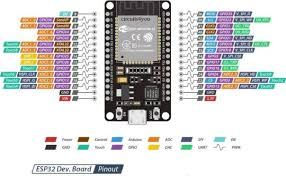
The project's implementation is characterized by robustness and flexibility. Code logic is meticulously crafted to synchronize actions between MQTT and SinricPro, ensuring consistent state updates and preempting conflicts or redundant actions. Furthermore, the utilization of non-blocking delays and update flags optimizes system responsiveness and resource utilization, fostering a smooth user experience.

This integration heralds a new era of IoT device interoperability and accessibility. By combining the ubiquity and simplicity of MQTT with the sophistication of SinricPro, users can seamlessly control and monitor IoT devices from anywhere, enhancing convenience, efficiency, and functionality across diverse domains. Furthermore, the project's scalability and extensibility empower developers to innovate and tailor solutions to meet evolving IoT requirements, thus unlocking a myriad of possibilities for transformative IoT applications.

**COMPONENTS:**

**ESP32:** The ESP32 is a powerful microcontroller with integrated Wi-Fi and Bluetooth capabilities, making it ideal for IoT applications. It features a dual-core processor, abundant GPIO pins, and support for various communication protocols. Its versatility and low cost make it a popular choice for both hobbyist and professional projects.

4o



**CODE:**

/#ifdef ESP8266

#include <ESP8266WiFi.h> /\* WiFi library for ESP8266 \*/

#else

#include <WiFi.h> /\* WiFi library for ESP32 \*/

#endif

#include <MQTT.h>

#include <Arduino.h>

#include "SinricPro.h"

#include "SinricProSwitch.h"

// WiFi credentials

#define WIFI\_SSID "moto g54 5G\_3480"

#define WIFI\_PASS "9500697232"

// Sinric Pro credentials

#define APP\_KEY "0dbfaadc-2e5b-447b-b59d-4f453e288beb"

#define APP\_SECRET "50a44a49-a614-47f3-8591-949a4059ccfd-5338ffda-063b-4b5f-a2a3-0540650ce907"

#define DEVICE\_ID "66644662888aa7f7a2354536"

// MQTT credentials

const char mqttUsername[] = "yellowroarer326";

const char mqttPassword[] = "jrSOKfZYGNgdnS7a";

// Pin definitions

#define LED\_PIN 2 // D2 (built-in LED for most ESP32 boards)

// MQTT client

WiFiClient net;

MQTTClient client(256); // Buffer size can be adjusted if needed

// MQTT topics

const char\* ledControlTopics = "/led/control";

const char\* ledStatusTopics = "/led/status";

// State tracking to avoid unnecessary updates

bool ledState = false;

bool updateFromMQTT = false;

bool updateFromSinricPro = false;

void connectWiFi() {

Serial.print("Connecting to WiFi...");

WiFi.begin(WIFI\_SSID, WIFI\_PASS);

while (WiFi.status() != WL\_CONNECTED) {

Serial.print(".");

delay(1000);

}

Serial.printf(" connected!\n[WiFi]: IP-Address is %s\n", WiFi.localIP().toString().c\_str());

}

void connectMQTT() {

Serial.print("Connecting to MQTT...");

while (!client.connect("ESP32Client", mqttUsername, mqttPassword)) {

Serial.print(".");

delay(1000);

}

Serial.println(" Connected to MQTT!");

client.subscribe(ledControlTopics);

}

void messageReceived(String &topic, String &payload) {

Serial.println("Incoming: " + topic + " - " + payload);

if (topic == ledControlTopics) {

if (payload == "b1 - on") {

ledState = true;

updateFromMQTT = true;

digitalWrite(LED\_PIN, HIGH); // Turn on LED

client.publish(ledStatusTopics, "B1 ON");

SinricProSwitch& mySwitch = SinricPro[DEVICE\_ID];

mySwitch.sendPowerStateEvent(ledState);

} else if (payload == "b1 - off") {

ledState = false;

updateFromMQTT = true;

digitalWrite(LED\_PIN, LOW); // Turn off LED

client.publish(ledStatusTopics, "B1 OFF");

SinricProSwitch& mySwitch = SinricPro[DEVICE\_ID];

mySwitch.sendPowerStateEvent(ledState);

}

updateFromMQTT = false;

}

}

bool onPowerState(const String &deviceId, bool &state) {

Serial.printf("Device %s turned %s\n", deviceId.c\_str(), state ? "on" : "off");

ledState = state;

updateFromSinricPro = true;

digitalWrite(LED\_PIN, state ? HIGH : LOW); // Turn LED on or off

client.publish(ledStatusTopics, state ? "B1 ON" : "B1 OFF");

updateFromSinricPro = false;

return true;

}

void setup() {

Serial.begin(9600); // Use a higher baud rate for better debugging output

// Connect to WiFi

connectWiFi();

// Initialize LED pin

pinMode(LED\_PIN, OUTPUT); // Set LED pin as output

digitalWrite(LED\_PIN, LOW); // Initialize LED to be off

// Setup SinricPro

Serial.println("Setting up SinricPro...");

SinricProSwitch& mySwitch = SinricPro[DEVICE\_ID];

mySwitch.onPowerState(onPowerState);

SinricPro.onConnected([]() {

Serial.println("Connected to SinricPro!");

});

SinricPro.onDisconnected([]() {

Serial.println("Disconnected from SinricPro!");

});

SinricPro.begin(APP\_KEY, APP\_SECRET);

SinricPro.restoreDeviceStates(true);

// Setup MQTT client

Serial.println("Setting up MQTT...");

client.begin("yellowroarer326.cloud.shiftr.io", 1883, net);

client.onMessage(messageReceived);

// Connect to MQTT

connectMQTT();

}

void loop() {

// Handle MQTT client

client.loop();

// Handle SinricPro

SinricPro.handle();

// Reconnect to WiFi if disconnected

if (WiFi.status() != WL\_CONNECTED) {

connectWiFi();

}

// Reconnect to MQTT if disconnected

if (!client.connected()) {

connectMQTT();

}

// Non-blocking delay

delay(100); // Small delay to yield to other processes

}

**CODE EXPLANATION:**

**Initialization:**

* At the beginning, the code initializes necessary libraries and defines credentials for WiFi, MQTT, and SinricPro.
* It also defines global variables to keep track of the LED state and flags to indicate the source of updates.

**Connection Functions:**

* **connectWiFi():** This function attempts to connect to the specified WiFi network and waits until the connection is established.
* **connectMQTT():** This function connects to the MQTT broker using the provided credentials.

**Message Handling:**

* **messageReceived():** This function is called whenever a message is received on MQTT. It checks if the message is meant to control the LED. If so, it updates the ledState, publishes the LED status, and notifies SinricPro about the state change.

**SinricPro Event Handling:**

* **onPowerState():** This function is a callback triggered when the power state changes in SinricPro. It updates the ledState, publishes the LED status, and ensures that the update doesn't cause unnecessary loops by checking the update flags.

**Setup Function:**

* In the setup() function, serial communication is initialized for debugging purposes.
* It connects to WiFi, sets up the LED pin, initializes SinricPro, and sets up the MQTT client.

**Main Loop:**

* In the main loop (loop()), the code continuously checks for MQTT messages, handles SinricPro events, and ensures WiFi and MQTT connections are maintained.
* The non-blocking delay allows other tasks to run smoothly without pausing the execution for a long time.

**How It Works:**

* When an MQTT message is received to turn the LED on or off, the code updates the ledState, publishes the LED status, and notifies SinricPro about the state change.
* When a state change is received from SinricPro, the code updates the ledState, publishes the LED status, and ensures that the update doesn't cause unnecessary loops.
* By tracking the source of updates with flags, the code prevents redundant updates and ensures that both MQTT and SinricPro stay synchronized without conflicts.

**STEPS TO CREATE A PROJECT**

**Sinric Pro :**

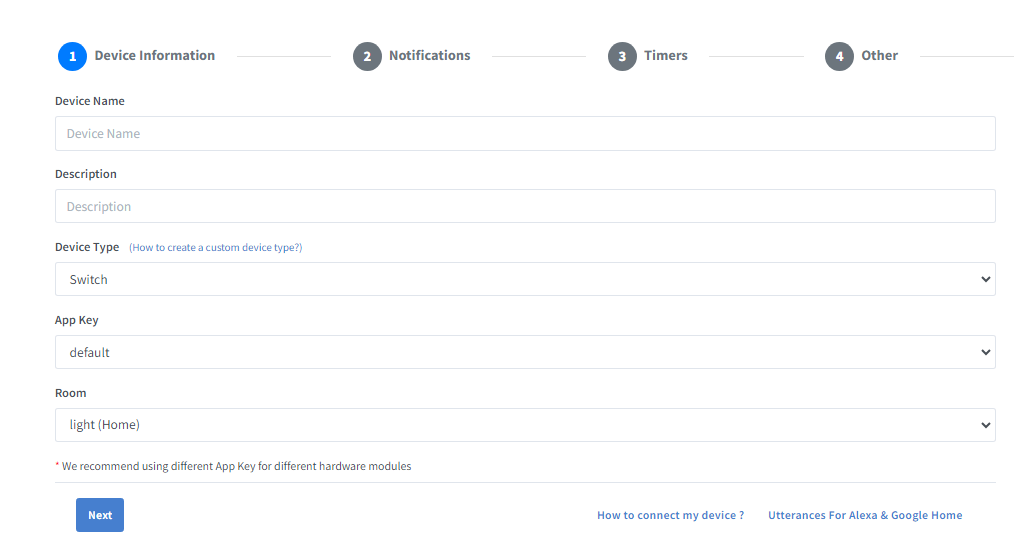
**1. Set Up Sinric Pro Account and Create a Device**

**Sign Up / Log In to Sinric Pro:**

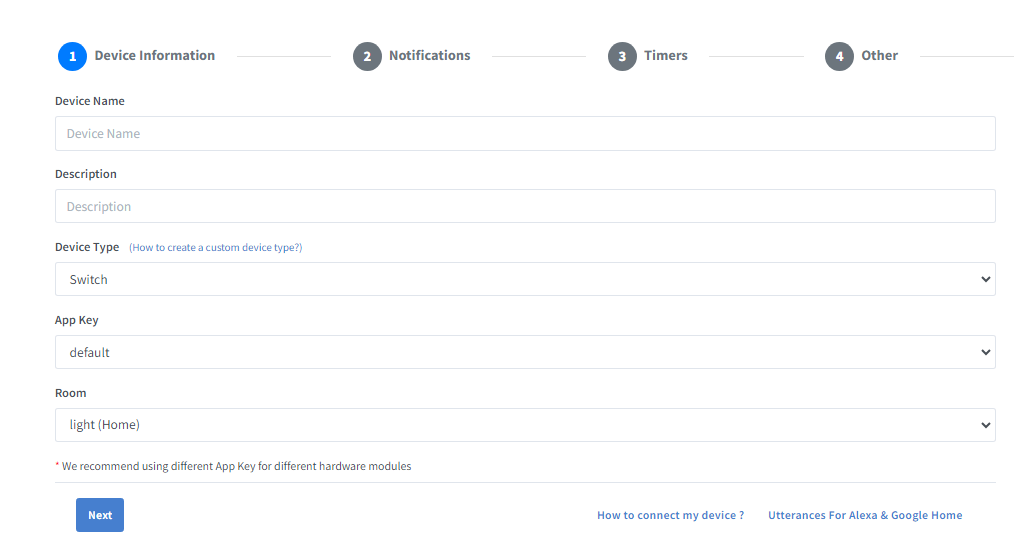
* Go to Sinric Pro and sign up for an account if you don't have one. Log in if you already have an account. (https://sinric.pro/).

**Create a New Device:**

* Once logged in, go to the Devices tab.
* Click on "Add Device".

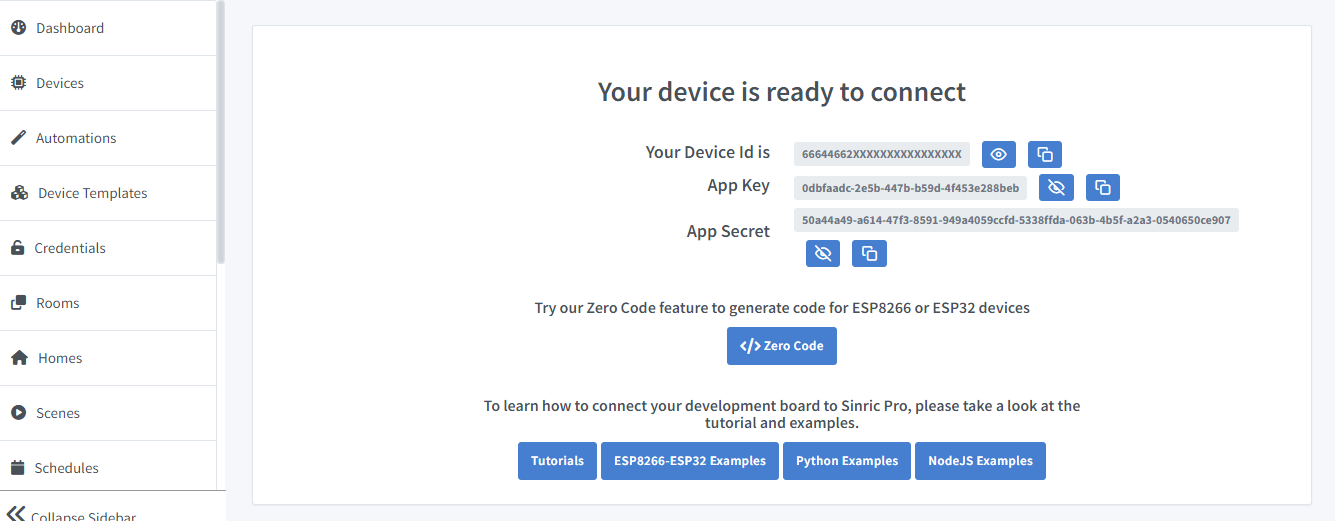


* Select "Switch" from the list of device types.
* Give your device a name (e.g., "My Light") and click "Save".



**Get Your Device Credentials:**

* After creating the device, you will see a list of your devices. Click on the "Switch" device you just created.
* Note down the Device ID.
* Click on "Credentials" in the sidebar to get your App Key and App Secret.



**2. Set Up the Hardware**

**Gather Components:**

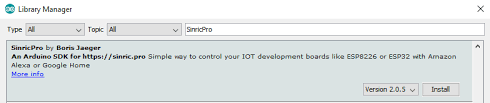
* ESP32 development board
* LED
* 220-ohm resistor
* Breadboard and jumper wires
* Connect the LED:
* Connect the anode (long leg) of the LED to a GPIO pin (e.g., GPIO 2).
* Connect the cathode (short leg) of the LED to one end of the resistor.
* Connect the other end of the resistor to the GND pin of the ESP32.

**Install Required Libraries:**

* In the Arduino IDE, go to Sketch > Include Library > Manage Libraries.
* Search for and install the following libraries:

SinricPro

WiFi



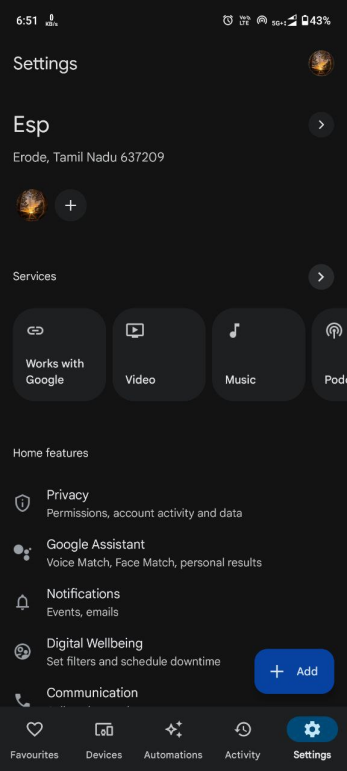
**Google HOME:**

**Open Google Home App:**

* On your smartphone, open the Google Home app.

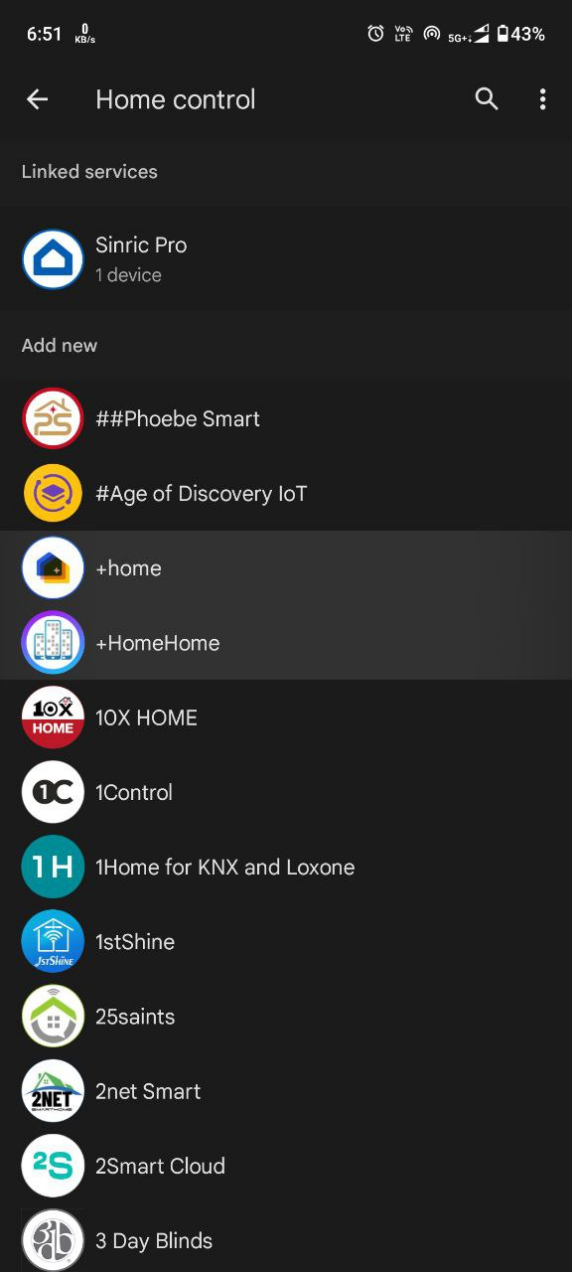
**Add a New Device:**

* Tap the "+" icon in the top left corner.
* Select "Set up device".
* Choose "Works with Google".



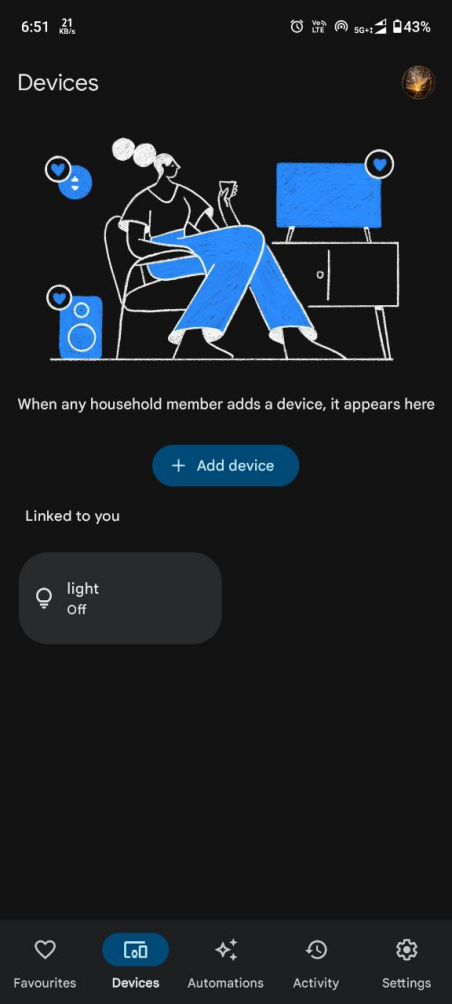
**Link Sinric Pro Account:**

* In the search bar, type "Sinric Pro" and select it from the list.
* Follow the on-screen instructions to link your Sinric Pro account with Google Home.
* Log in with your Sinric Pro credentials and authorize Google to access your Sinric Pro devices.



**Assign Devices to Rooms (Optional):**

* Once linked, you can assign the device (e.g., "My Light") to a room in your house for better organization.



**Test with Google Assistant:**

* You can now use Google Assistant to control the light.
* Say "Hey Google, turn on [device name]" or "Hey Google, turn off [device name]".

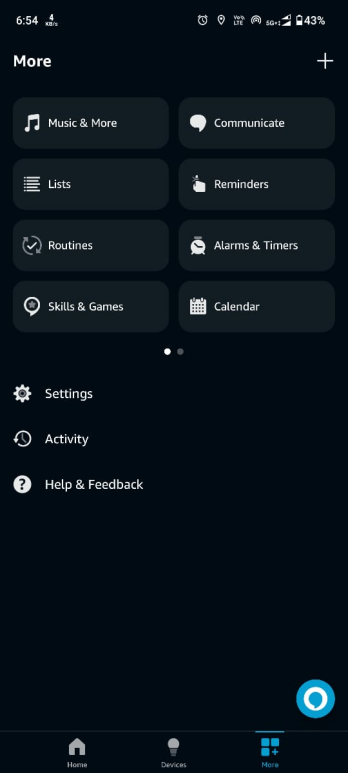
**ALEXA:**

**Open Alexa App:**

* On your smartphone, open the Amazon Alexa app.

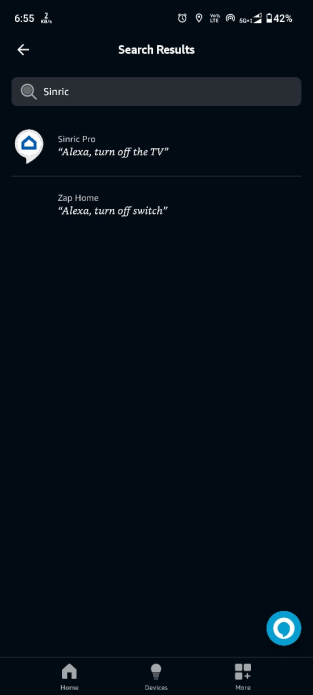
**Enable Sinric Pro Skill:**

* Tap on the "More" button in the bottom right corner.
* Select "Skills & Games".
* Tap the search icon and type "Sinric Pro".
* Select "Sinric Pro" from the list and tap "Enable to Use".



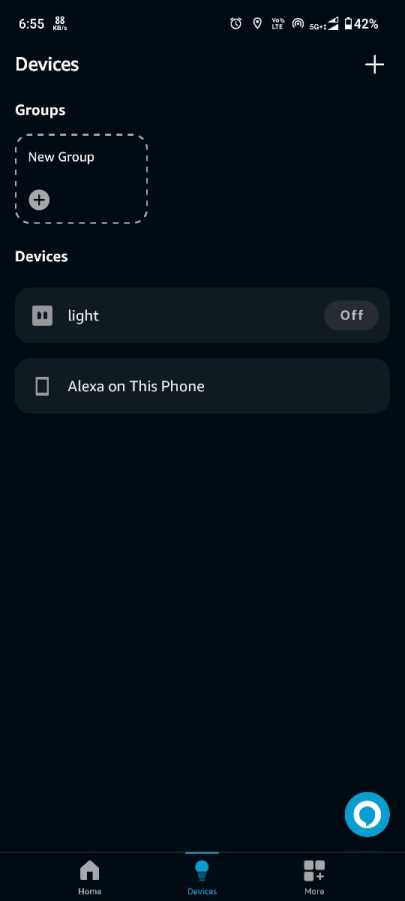
**Link Sinric Pro Account:**

* Log in with your Sinric Pro credentials and authorize Alexa to access your Sinric Pro devices.



**Discover Devices:**

* After linking, Alexa will automatically discover your Sinric Pro devices.
* If not, you can manually trigger discovery by saying "Alexa, discover devices" or by going to the Devices tab in the Alexa app and selecting "Add Device".



**Assign Devices to Groups (Optional):**

* You can create groups in the Alexa app and add your device to a group for easier control.

**Test with Alexa:**

* You can now use Alexa to control the light.
* Say "Alexa, turn on [device name]" or "Alexa, turn off [device name]".

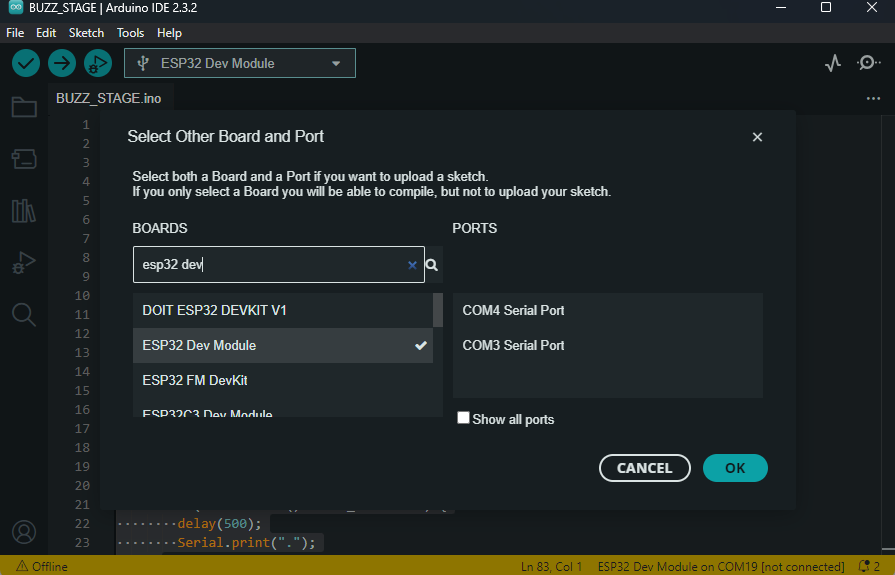
**MQTT:**

**Install Arduino IDE:**

* Download and install the Arduino IDE from the official website (https://www.arduino.cc/en/software).

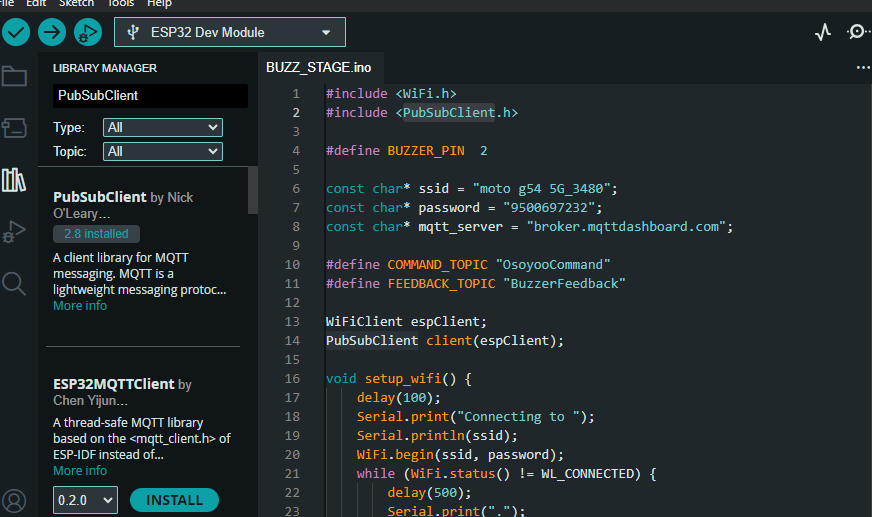
**Install ESP32 Board in Arduino IDE:**

* Open the Arduino IDE.
* Go to Tools > Board > Boards Manager.
* Search for ESP32 and install the esp32 by Espressif Systems.



**Install PubSubClient Library:**

* In the Arduino IDE, go to Sketch > Include Library > Manage Libraries.
* Search for PubSubClient and install it.

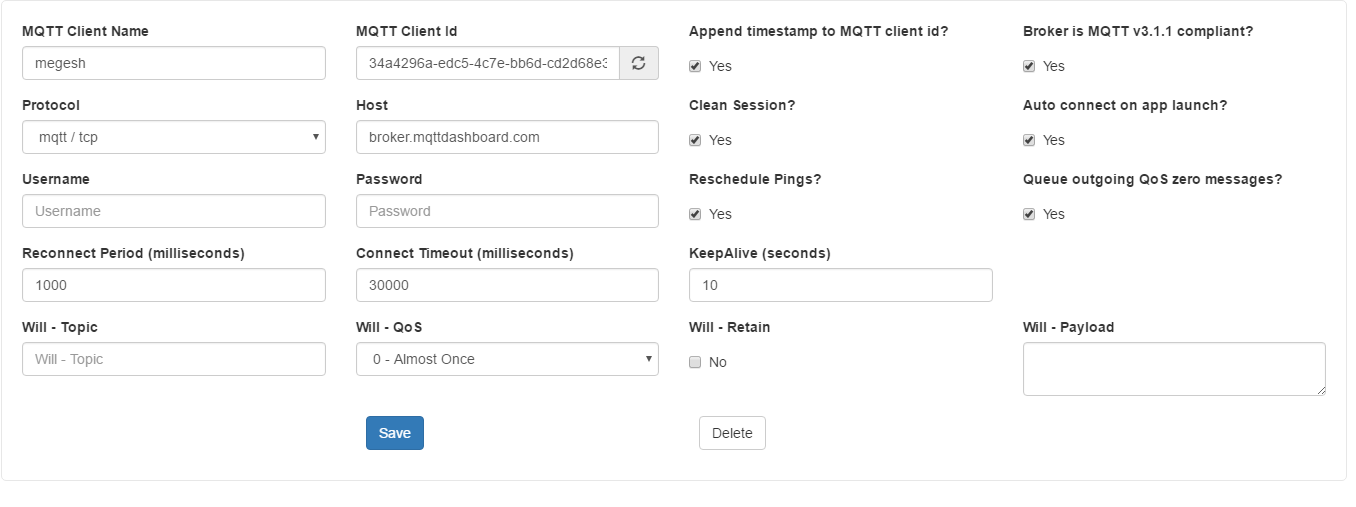


**Write the Code:**

* Open a new sketch in Arduino IDE and copy the following code:
* Connect your ESP32 to your computer using a USB cable.
* Select the correct board and port in the Arduino IDE:
* Go to Tools > Board > ESP32 Dev Module.
* Go to Tools > Port and select the COM port to which the ESP32 is connected.
* Click on the upload button to compile and upload the code to the ESP32.

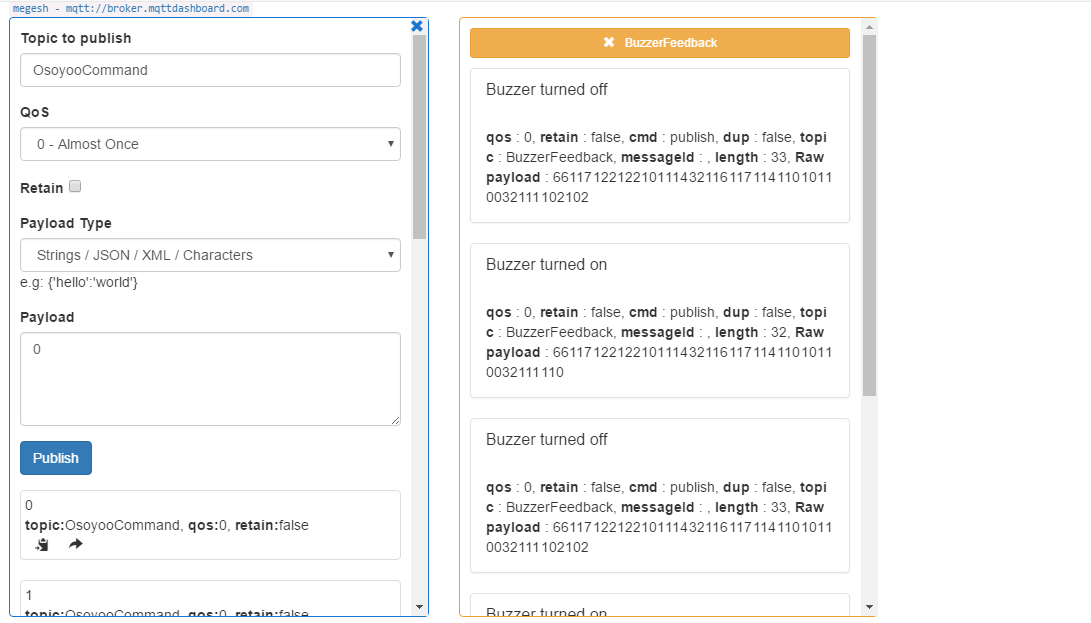
**Set Up MQTT Box:**

* Download and install MQTT Box (available as a browser extension or standalone application).
* Open MQTT Box and create a new MQTT client.
* Configure the client with the following settings:
* Client ID: Any unique identifier (e.g., MQTTBoxClient).
* Protocol: MQTT/TCP.
* Host: broker.mqttdashboard.com.
* Port: 1883.
* Connect the client.

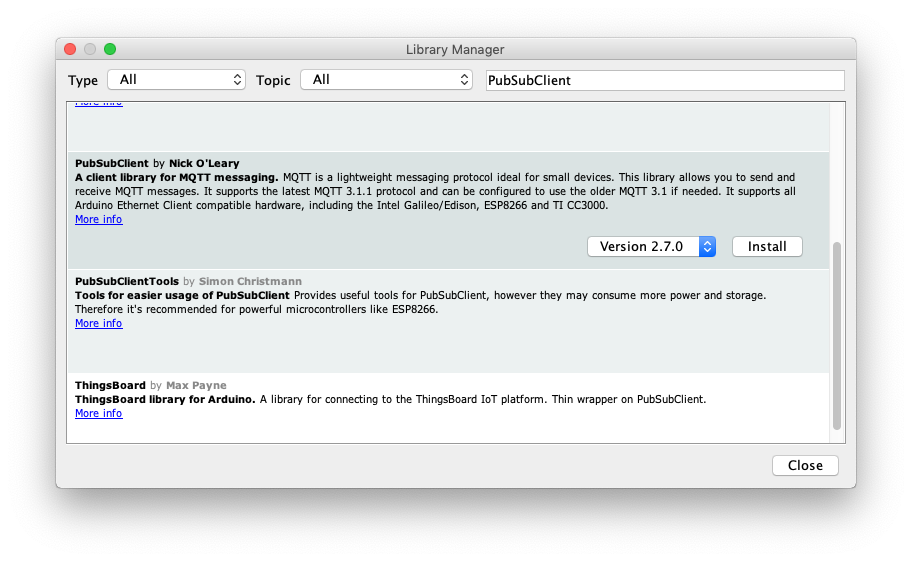


**Send Commands Using MQTT Box:**

* Subscribe to the feedback topic (BuzzerFeedback) to see the buzzer status.
* Publish messages to the command topic (OsoyooCommand):
* To turn the buzzer on, publish the message 1 to OsoyooCommand.
* To turn the buzzer off, publish the message 0 to OsoyooCommand.
* Observe the feedback messages in the subscribed feedback topic to confirm the buzzer status.



**PubSubClient:**



The PubSubClient library is a lightweight and efficient library designed for Arduino-compatible boards to facilitate MQTT communication. It allows these microcontrollers, such as the ESP32, to connect to an MQTT broker, publish messages to topics, and subscribe to topics to receive messages. This library is crucial for implementing IoT applications, enabling seamless and reliable data exchange between devices over the MQTT protocol.

**CONCLUSION:**

In conclusion, this project exemplifies the convergence of IoT technologies, showcasing the effective integration of an ESP32-based IoT device with MQTT and SinricPro. By leveraging MQTT for real-time communication and SinricPro for cloud-based control and integration with smart home platforms, the project achieves a harmonious synergy that enhances the device's utility and accessibility.

Through meticulous software development and configuration, the IoT device becomes a versatile tool for remote control and monitoring. Users can effortlessly manipulate the device's state using MQTT messages or voice commands, seamlessly integrating it into their smart home ecosystems or industrial automation setups.

The project's success underscores the importance of interoperability and flexibility in IoT deployments. By embracing open standards like MQTT and leveraging cloud services such as SinricPro, developers can create robust and scalable solutions that adapt to diverse use cases and environments.

Looking ahead, the lessons learned from this project pave the way for further innovation and exploration in the realm of IoT. Future endeavors may focus on enhancing security measures, optimizing resource utilization, and expanding compatibility with additional protocols and platforms, thereby enriching the IoT landscape and empowering users with ever more sophisticated and seamless experiences.