Integration of Tektork IoT kit with MQTT and MQTT Explorer

1. Introduction

1.1. Overview of ESP32 in IoT Systems

The ESP32 is a versatile and powerful microcontroller featuring integrated Wi-Fi and Bluetooth capabilities, making it a popular choice for Internet of Things (IoT) applications. It supports a broad range of sensors and peripherals and provides robust wireless communication, which is essential for connecting embedded devices to cloud platforms and local servers. The ESP32's low power consumption, rich peripheral sets, and extensive software ecosystem enable developers to implement diverse IoT solutions, including data acquisition, device control, and remote monitoring. This microcontroller bridges the gap between physical sensors and IoT software frameworks such as MQTT, Node-RED, and cloud services like ThingSpeak and Firebase, facilitating seamless device-to-cloud integration and real-time data visualization.

1.2. Objective

This manual aims to guide users through the comprehensive setup, programming, and deployment of various IoT projects by interfacing Tektork IoT Kit with multiple popular platforms and technologies including ThingSpeak, MQTT brokers, Firebase cloud databases, Node-RED for flow-based programming, and the TIG Stack (Telegraf, InfluxDB, Grafana) for advanced data storage, monitoring and visualization. The objective is to provide a step-by-step reference combining hardware setup, firmware development, platform configuration, and dashboard creation, enabling readers to design scalable, networked IoT systems from scratch with detailed technical insights and best practices.

1.3. Required Hardware and Software

Hardware Requirements

For this manual and related projects, the hardware setup is kept minimal and focused on essential components:

- Tektork IoT Kit: The core microcontroller module with built-in Wi-Fi and Bluetooth connectivity, serving as the primary IoT device.
- USB Type-C Cable: Used to both power the ESP32 board and program it from a PC or laptop.
- PC or Laptop: Required for software development, programming, and interfacing with cloud platforms and local systems.

This simplified hardware setup optimizes project accessibility, making it feasible for users to start IoT development with the Tektork IoT Kit, connecting cable, and a host computer without additional peripheral sensors or modules.
Software Requirements:
The software environment consists of tools and platforms necessary for programming the ESP32, managing MQTT communications, cloud data aggregation, and IoT visualization:

- Arduino IDE or ESP-IDF: Primary development environments for writing and uploading firmware to the ESP32 microcontroller.
- MQTT Broker Service: Public brokers like HiveMQ or Mosquitto provide the messaging backbone for IoT device communication.
- ThingSpeak Account: A cloud-based IoT data platform for acquiring, storing, and visualizing data from connected devices.
- Firebase Console: Google's cloud service offering a real-time database, authentication, and hosting for IoT backend needs.
- Node-RED: A visual flow-based programming tool for creating IoT workflows and dashboards without coding complexity.
- TIG Stack Components:
  Telegraf: An agent for collecting and processing time-series data.
  InfluxDB: A time-series optimized database for storing sensor data.
  Grafana: A dashboard platform to visualize and analyze IoT data effectively.
- Supporting Libraries and Tools: MQTT client libraries for ESP32, HTTP client libraries, and SDKs compatible with ESP32 and relevant platforms.

This software setup targets versatility and scalability, enabling development from basic sensor data visualization up to advanced real-time monitoring and control systems.

2. Setting Up the Tektork IoT Kit Development Environment

Refer the Document tilted
a) Tektork-Kit-ArduinoIDE-Driver-Installation-Notes.pdf
b) Tektork-IoT-Board-Hardware-Manual-V1.0.pdf

3. MQTT with ESP32

3.1. Introduction to MQTT Protocol

MQTT (Message Queuing Telemetry Transport) is a lightweight, publish-subscribe messaging protocol ideal for IoT devices like the ESP32.

- It operates over TCP/IP, optimizing low bandwidth and power consumption.
- Devices (clients) either publish data (sensor readings, commands) or subscribe to specific topics to receive updates.
- A central MQTT broker manages these message exchanges, ensuring decoupled communication between devices.

3.2. Installing and Using Public MQTT Broker(HiveMQ/Mosquitto)

Public brokers such as HiveMQ (broker.hivemq.com) and Mosquitto (test.mosquitto.org) can be used without local installation:

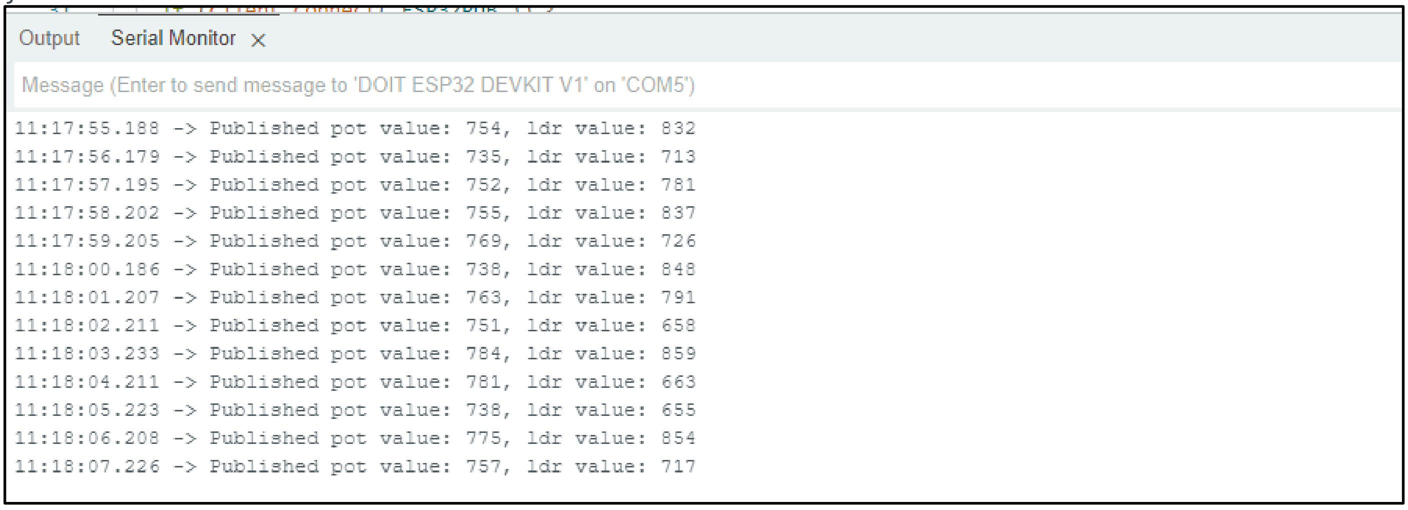
- HiveMQ Example:
  - Broker Address: broker.hivemq.com
  - Port: 1883
- Mosquitto Example:
  - Broker Address: test.mosquitto.org
  - Port: 1883
- Use any MQTT client (like MQTT Explorer, Node-RED, or online HiveMQ Web Client) to test publishing/subscribing to topics like esp32led, esp32pot, etc.
- No registration or authentication is required for public brokers, but never use them for sensitive data.

3.3. ESP32 MQTT Publisher Code(Sensor Data to Broker)

The ESP32 reads sensor values (e.g., potentiometer/POT on GPIO 36, LDR on GPIO 39), connects to Wi-Fi, and publishes readings to MQTT topics.

Here is a typical code structure, following your documented style:

```
#include <WiFi.h>
#include <PubSubClient.h>

const char* ssid = "S23FE";
const char* password = "Sanjey123";
const char* mqttserver = "broker.hivemq.com";
const int mqttport = 1883;

const char* topicpot = "esp32pot";
const char* topicldr = "esp32ldr";

#define POTPIN 36     // Potentiometer
#define LDRPIN 39     // LDR

WiFiClient espClient;
PubSubClient client(espClient);

void setupWiFi() {
  Serial.print("Connecting to WiFi");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println(" connected");
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP32PUB")) {
      Serial.println("connected");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

void setup() {
  Serial.begin(115200);
  setupWiFi();
  client.setServer(mqttserver, mqttport);
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  int potValue = analogRead(POTPIN); // 0-4095
  int ldrValue = analogRead(LDRPIN); // 0-4095

  char buffer[16];
  snprintf(buffer, sizeof(buffer), "%d", potValue);
  client.publish(topicpot, buffer);

  snprintf(buffer, sizeof(buffer), "%d", ldrValue);
  client.publish(topicldr, buffer);

  Serial.print("Published pot value: ");
  Serial.print(potValue);
  Serial.print(", ldr value: ");
  Serial.println(ldrValue);

  delay(1000); // Update every second
}
```


Output
Serial Monitor
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM5')

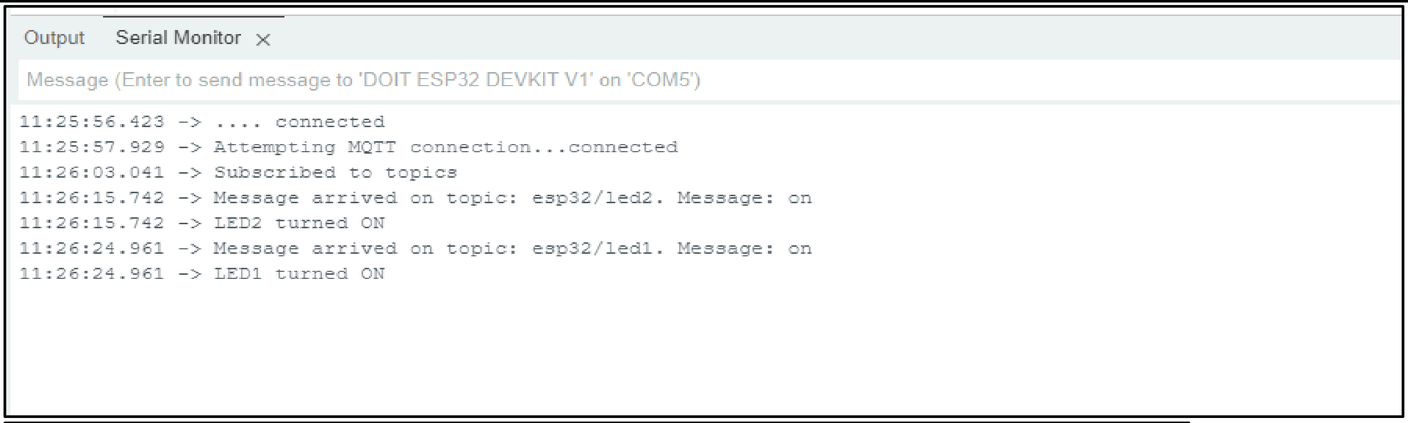
```
11:17:55.188 -> Published pot value: 754, ldr value: 832
11:17:56.179 -> Published pot value: 735, ldr value: 713
11:17:57.195 -> Published pot value: 752, ldr value: 781
11:17:58.202 -> Published pot value: 755, ldr value: 837
11:17:59.205 -> Published pot value: 769, ldr value: 726
11:18:00.186 -> Published pot value: 738, ldr value: 848
11:18:01.207 -> Published pot value: 763, ldr value: 791
11:18:02.211 -> Published pot value: 751, ldr value: 658
11:18:03.233 -> Published pot value: 784, ldr value: 859
11:18:04.211 -> Published pot value: 781, ldr value: 663
11:18:05.223 -> Published pot value: 738, ldr value: 655
11:18:06.208 -> Published pot value: 775, ldr value: 854
11:18:07.226 -> Published pot value: 757, ldr value: 717
```

This code connects ESP32 to WiFi, then publishes potentiometer and LDR readings to MQTT topics every second.

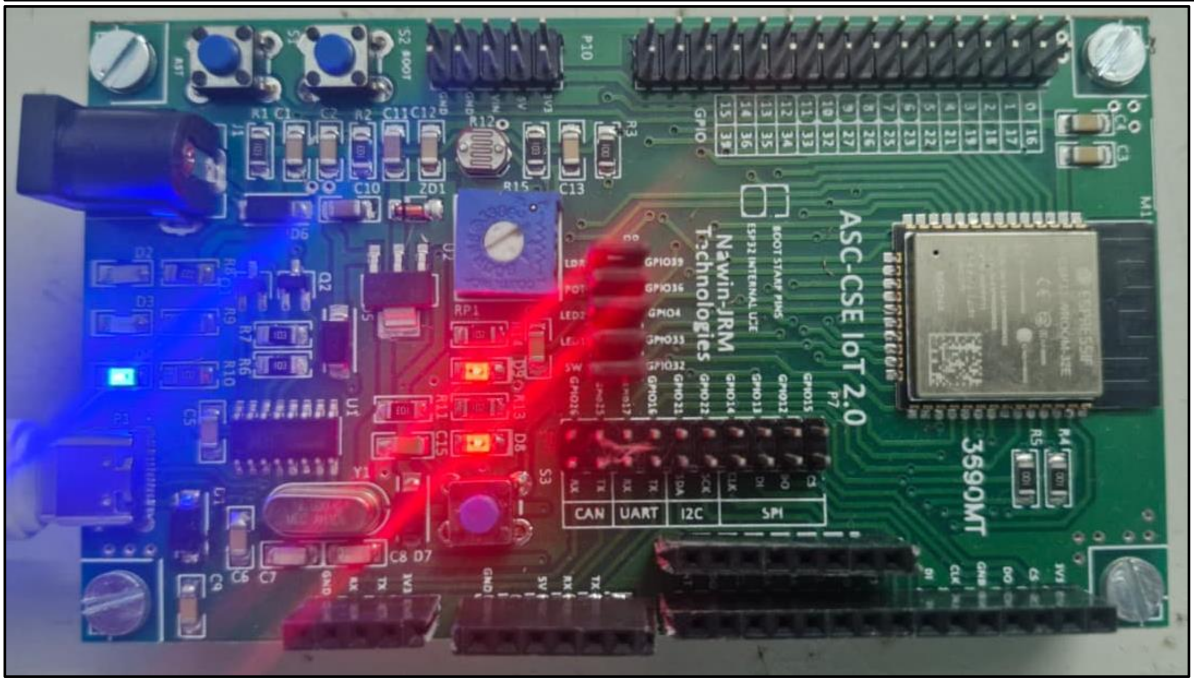
3.4. ESP32 MQTT Subscriber Code(LED/Switch Control)

ESP32 subscribes to topics that control LEDs (GPIO 33, GPIO 4). It reacts to "on"/"off" string payloads from the broker and switches the LEDs accordingly:

```
#include <WiFi.h>
#include <PubSubClient.h>

const char* ssid = "S23FE";
const char* password = "Sanjey123@";
const char* mqttserver = "broker.hivemq.com";
const int mqttport = 1883;

const char* topicled1 = "esp32/led1";
const char* topicled2 = "esp32/led2";

#define LED1PIN 33
#define LED2PIN 4

WiFiClient espClient;
PubSubClient client(espClient);

void setupWiFi() {
  Serial.print("Connecting to WiFi");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println(" connected");
}

void callback(char* topic, byte* payload, unsigned int length) {
  String msg;
  for (unsigned int i = 0; i < length; i++) {
    msg += (char)payload[i];
  }
  msg.trim();

  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  Serial.println(msg);

  if (String(topic) == topicled1) {
    if (msg == "on") {
      digitalWrite(LED1PIN, HIGH);
      Serial.println("LED1 turned ON");
    }
    else if (msg == "off") {
      digitalWrite(LED1PIN, LOW);
      Serial.println("LED1 turned OFF");
    }
  }
  else if (String(topic) == topicled2) {
    if (msg == "on") {
      digitalWrite(LED2PIN, HIGH);
      Serial.println("LED2 turned ON");
    }
    else if (msg == "off") {
      digitalWrite(LED2PIN, LOW);
      Serial.println("LED2 turned OFF");
    }
  }
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Add random number to client ID to avoid collisions
    String clientId = "ESP32DualLEDClient-";
    clientId += String(random(0xffff), HEX);
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
      client.subscribe(topicled1);
      client.subscribe(topicled2);
      Serial.println("Subscribed to topics");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

void setup() {
  Serial.begin(115200);
  pinMode(LED1PIN, OUTPUT);
  pinMode(LED2PIN, OUTPUT);
  digitalWrite(LED1PIN, LOW);
  digitalWrite(LED2PIN, LOW);

  setupWiFi();

  client.setServer(mqttserver, mqttport);
  client.setCallback(callback);

  randomSeed(micros());
}

void loop() {
  if (!client.connected()) reconnect();
  client.loop();
}
```


Output
Serial Monitor
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM5')

```
11:25:56.423 -> .... connected
11:25:57.929 -> Attempting MQTT connection...connected
11:26:03.041 -> Subscribed to topics
11:26:15.742 -> Message arrived on topic: esp32/led2. Message: on
11:26:15.742 -> LED2 turned ON
11:26:24.961 -> Message arrived on topic: esp32/led1. Message: on
11:26:24.961 -> LED1 turned ON
```



This code listens for messages on two MQTT topics and switches LEDs accordingly.

3.5. Working Model Demonstration(Two way Communication)

Test Setup

- Publisher: ESP32 reads POT and LDR, publishes to topics esp32pot and esp32ldr.
- Subscriber: ESP32 subscribes to esp32led1 and esp32led2, controlling LEDs on GPIO 33, 4.
- Broker: Public HiveMQ or Mosquitto instance on port 1883.
- Dashboard: Use Node-RED, MQTT Explorer, or HiveMQ Web Client to visualize sensor data/topics and control LEDs.

4. MQTT Explorer

4.1. Introduction to MQTT Explorer Tool

MQTT Explorer is a free, desktop-based GUI client designed for debugging and visualizing MQTT traffic in real time.

- It presents MQTT messages in a clear, tree-structured format, showing all published topics and payloads.
- It is widely used for testing, debugging, and validating IoT devices such as ESP32 microcontrollers sending sensor data or receiving commands.

4.2. Installing MQTT Explorer on(Windows/Linux)

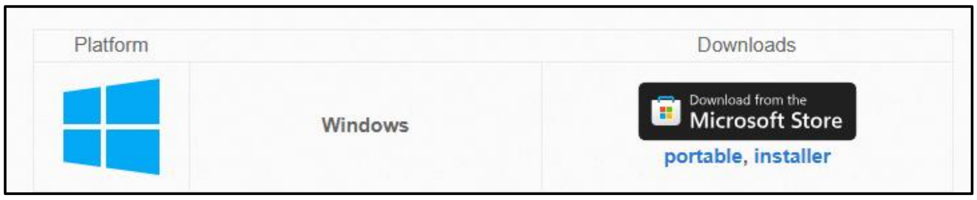
Windows

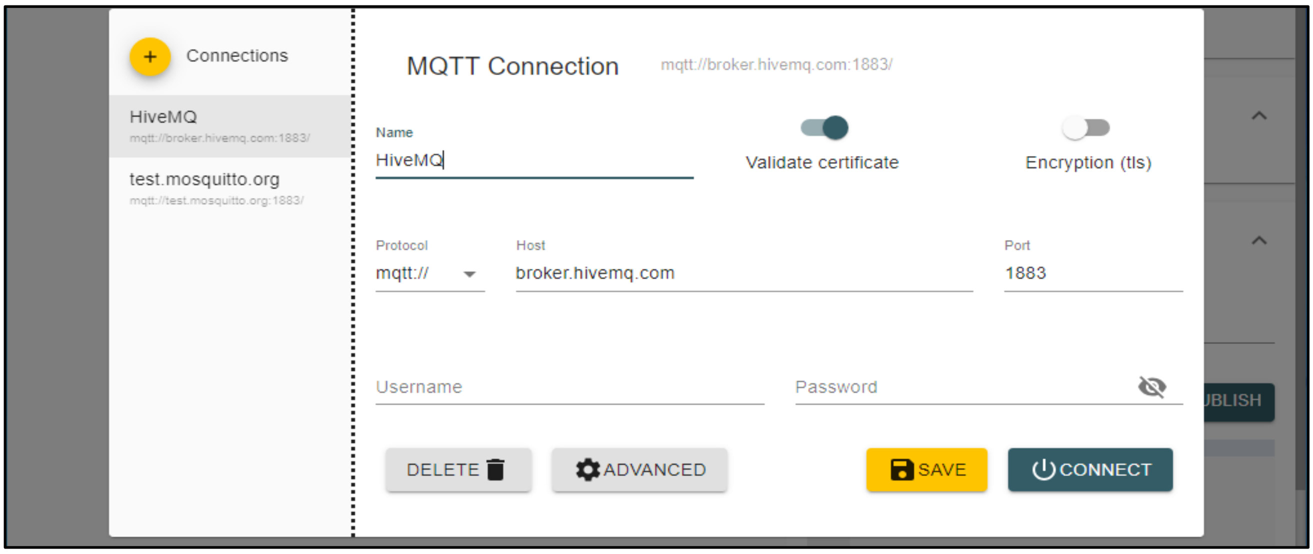
- Visit the official MQTT Explorer site (mqtt-explorer.com) and download the Windows installer (.exe).
- Run the installer by double-clicking the downloaded file.
- Complete the setup steps (Next, Install, Finish).
- A desktop shortcut will be created for quick access.

Linux

- Download the .AppImage from the official site or use Snap:
  - sudo snap install mqtt-explorer
- Make the AppImage executable:
  - chmod +x mqtt-explorer-*.AppImage
  - ./mqtt-explorer-*.AppImage

Both systems provide a simple interface after first launch, ready for broker connection.





MQTT Connection mqtt://broker.hivemq.com:1883/

HiveMQ mqtt://broker.hivemq.com:1883/

test.mosquitto.org mqtt://test.mosquitto.org:1883/

Name: HiveMQ

Validate certificate: on

Encryption (tls): off

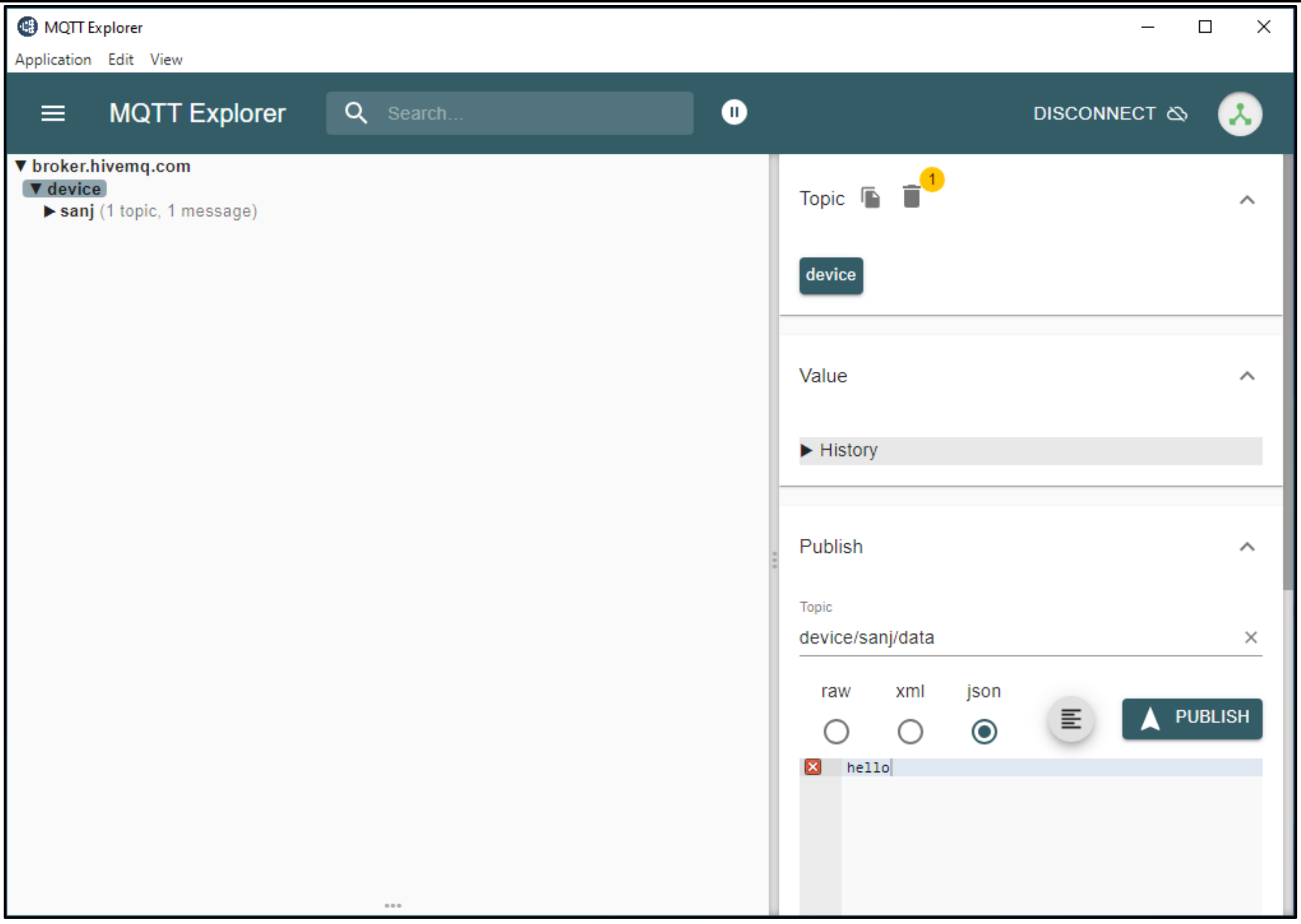
Protocol: mqtt://

Host: broker.hivemq.com

Port: 1883

Username

Password



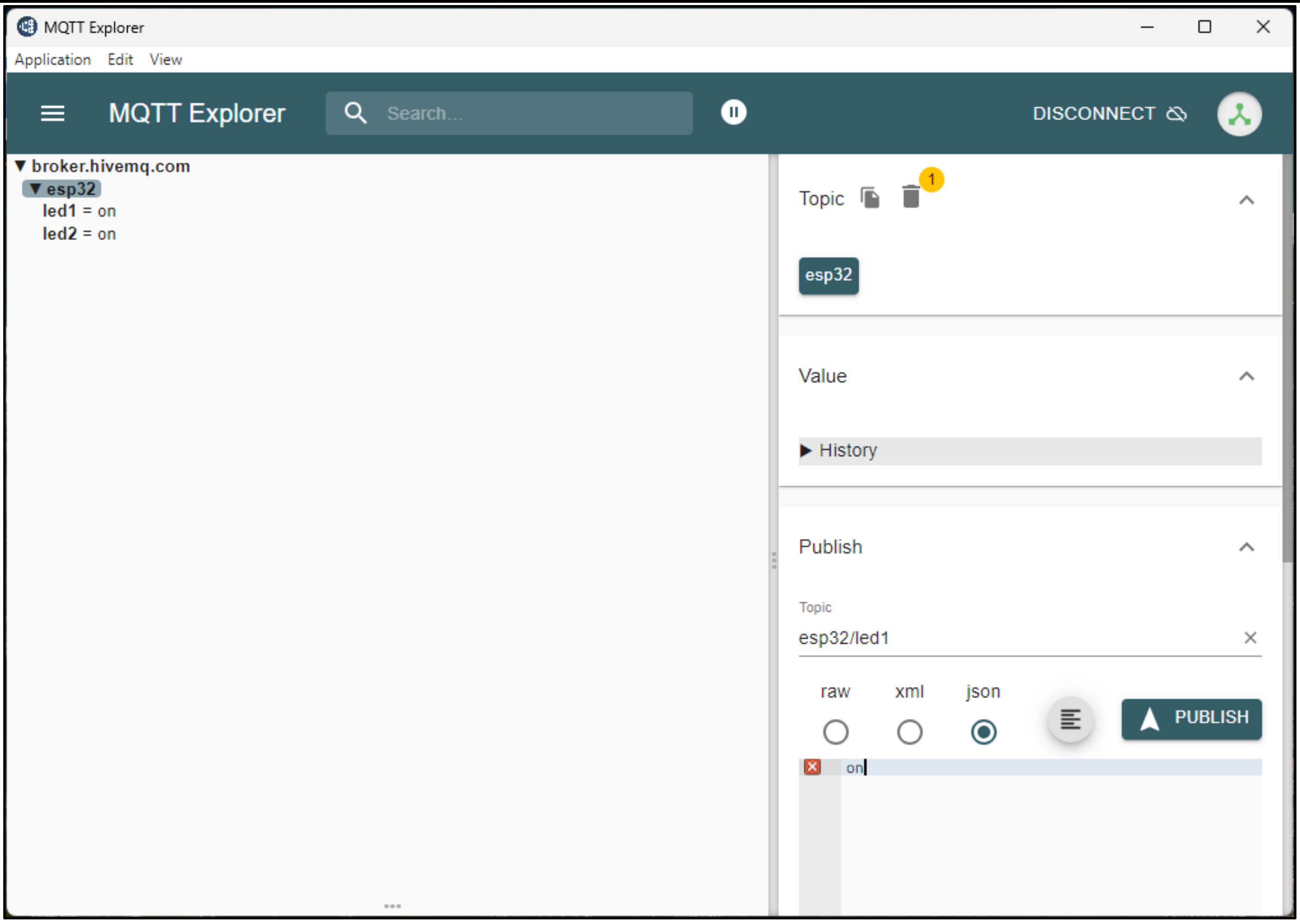
4.3. Connecting to MQTT Broker

- Open MQTT Explorer and click “Add New Connection.”
- Provide a descriptive name, such as "HiveMQ" or "Mosquitto".
- Enter the broker host (for public HiveMQ: broker.hivemq.com, for Mosquitto: test.mosquitto.org) and port number (1883).
- Save and click "Connect" to start the session.
- You can specify particular topics (e.g., esp32pot, esp32led1) for focused subscription during setup or add them later in the interface.

4.4. Monitoring ESP32 Published and Subscribed Topics

Once connected, MQTT Explorer offers live monitoring:

- Published topics (from ESP32):
  - esp32pot: Potentiometer sensor data
  - esp32ldr: LDR sensor data
- Subscribed topics (ESP32 listening):
  - esp32led1 and esp32led2: Remote LED control commands
- Click any topic to view the latest message payloads and history.
- The tool allows sending test payloads directly to any topic (for example, publish "on" or "off" to esp32led1 to toggle the LED).
- Changes made are instantly reflected both on the ESP32 device and the Explorer tree view, easing validation and debugging.



▼ broker.hivemq.com
▼ esp32
led1 = on
led2 = on

Topic: esp32

Publish Topic: esp32/led1

raw ( ) xml ( ) [✓ json]

on

4.5. Working Model Demonstration(ESP32 and MQTT Explorer Debugging)

End-to-End Debugging Flow

- ESP32 code publishes sensor values (esp32pot, esp32ldr) and listens for LED control messages (esp32led1, esp32led2).
- MQTT Explorer displays incoming sensor values in real time under their respective topics, verifying that the ESP32 is sending correct data.
- Use MQTT Explorer to send control messages to the LED topics; the ESP32 will immediately switch the LEDs on/off based on the message syntax ("on"/"off").
- This setup supports full two-way communication verification; both data publish and actuator control paths are visible, interactive, and logged for analysis.