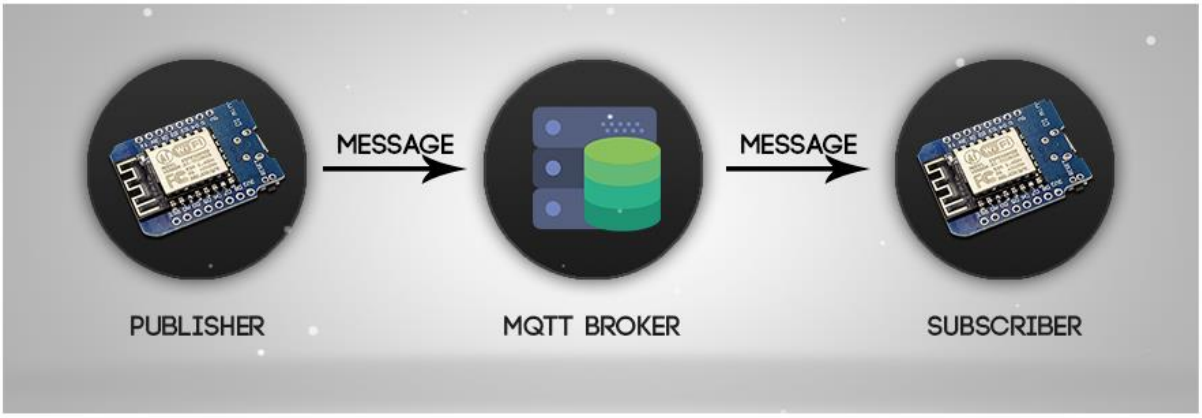
ESP8266 D1 Mini MQTT - Der Start mit MQTT

Beschreibung

Zum Austausch von IoT Daten im internen Netzwerk wir gerne das MQTT Protokoll verwendet. In dieser Beschreibung wird das Thema in der Praxis mit einem D1-mini gezeigt.

Internet Quelle: https://makesmart.net/esp8266-d1-mini-mqtt/



Was ist MQTT

MQTT steht für **Message Queuing Telemetry Transport** und ist ein Nachrichtenprotokoll, welches geringe Anforderungen an das Netzwerk und die Hardware stellt. Die Entwicklerseite beschreibt das Protokoll als ein machine-to-machine oder IoT Verbindungsprotokoll.

MQTT bietet ein sehr gutes Error-Handling, integriert QoS und bietet Funktionen um die Übertragung einer Nachricht zu versichern. Die Liste an Möglichkeiten ist groß. Deshalb werden wir uns zuerst mit den wichtigsten Dingen beschäftigen und einen Vergleich zu HTTP aufstellen.

HTTP vs MQTT

Wo besteht der Vorteil in der Nutzung von MQTT im Vergleich zu HTTP?

Wie oben bereits erwähnt, ist MQTT ein sehr schlankes Protokoll, welches gut im Bereich von Netzwerken mit wenig Bandbreite genutzt werden kann. Der kleinste mögliche MQTT-Header hat eine Größe von gerade mal 2 Byte. So ist der Overhead an Daten, die durch das Netzwerk fließen, im Gegensatz zur HTTP sehr gering.

Oft wird MQTT im Bereich von batteriebetriebenen Sensoren im IoT-Bereich eingesetzt, da das Protokoll wenig Ressourcenanforderungen an die Hardware stellt und es somit nur einen geringen Stromverbrauch gibt.

HTTP basiert auf einer Request-, Response-Architekturen (Client-Server-Modell). Dadurch ist es nicht ohne weiteres möglich eine Nachricht vom Server an den Client zu pushen. MQTT hingegen arbeitet

bidirektional und es können sowohl Anfragen an den Server gesendet als auch empfangen werden (ohne vorherigen Request).

Unter MQTT ist es möglich, im Gegensatz zur 1:1-Kommunikation von HTTP, eine Nachricht an mehrere Empfänger zu schicken (1:n). Der Empfänger muss zu dieser Zeit online sein.

Es gibt noch weitere Vorteile, wie z.B. die 3 Arten von QoS. Allerdings sollen diese Informationen für den Einstieg reichen.

Grundlegende Funktionsweise

MQTT ist ein Client-Server-Protokoll und basiert auf einer Publish / Subscribe Architektur. Der Server wird in diesem Fall **Broker** genannt und ist zuständig für die Koordination der Nachrichten. Der Broker empfängt Nachrichten und gibt diese an die richtigen Empfänger weiter. Jede Nachricht in einer MQTT-Kommunikation enthält eine **Topic**, also quasi ein Thema. Die Topics sind hierarchisch, wie in einer Baumstruktur, aufgebaut.

Hierzu ein Beispiel:

```
makesmart/forum/user/findo1
makesmart/forum/artikel/id5
```

Im Makesmart-Forum gibt es also verschiedene Benutzer und verschiedene Artikel. Ein Client kann nun an eine dieser Topics eine Nachricht schicken. Unter MQTT nennt man das eine **PUBLISH**. Alle Clients die dieses Topic nun **SUBSCRIBED** haben empfangen die Nachricht.

Außerdem ist es möglich Wilcards zu SUBSCRIBEN.

- *makesmart/forum/artikel/#* bedeutet, dass der Client alle Artikel aus dem makesmart-Forum empfängt.
- *makesmart/+/artikel/id5* bedeutet, dass der Client alle Artikel mit der ID 5 aus allen verfügbaren makesmart-Foren empfängt.

Integration im ESP8266 D1 Mini

Um MQTT in Kombination mit dem ESP8266 nutzen zu können muss in der Arduino IDE zuerst eine passende MQTT-Libary eingebunden werden. Hier gibt es eine große Auswahl. Ich nutze für diesen Artikel die **PubSubClient-Libary**, die ihr einfach über den Bibliotheksverwalter Werkzeuge > Bibliotheken verwalten... hinzufügen könnt.

Die hinzugefügte Library wird oben im eurem Quellcode mit einer Zeile eingebunden.

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

Verbindung zu einem WLAN und MQTT-Broker

```
1  #include <ESP8266WiFi.h>
2  #include <PubSubClient.h>
3
4  const char* SSID = "WIFI_SSID";
5  const char* PSK = "WIFI_KEY";
6  const char* MQTT_BROKER = "test.mosquitto.org";
7
8  WiFiClient espClient;
9  PubSubClient client(espClient);
10
11 void setup() {
12     Serial.begin(115200);
13     setup_wifi();
14     client.setServer(MQTT_BROKER, 1883);
15 }
16
17 void setup_wifi() {
18     WiFi.begin(SSID, PSK);
19
20     while (WiFi.status() != WL_CONNECTED) {
21         delay(100);
22     }
23
24     Serial.println(WiFi.localIP());
25 }
26 void loop() {
27     if (!client.connected()) {
28         while (!client.connected()) {
29             client.connect("ESP8266Client");
30             delay(100);
31         }
32     }
33     client.loop();
34 }
```

Um eine Verbindung mit eurem WLAN-Netzwerk zu ermöglichen, müssen Zeile 4 und 5 angepasst werden.

Erklärung des Codes

In der Setup-Schleife wird dann definiert, zu welchem Broker sich der Client verbinden soll und unter welchen Port er dies machen soll. Sobald eine WIFI-Verbindung vom ESP zu eurem WLAN hergestellt wurde und die Loop-Schleife ausgeführt wird, beginnt der eigentliche Verbindungsaufbau zum Broker.

Erst wird überprüft, ob der Client schon verbunden ist. Wenn er das nicht ist, dann wird versucht eine Verbindung aufzubauen, solange bis diese steht. Beim nächsten Durchgang der Loop-Schleife wird dieser Teil dann ignoriert, sofern die Verbindung noch aktiv ist.

Publish (Publizieren, senden)

Der PUBLISH ist sehr simple. Dazu fügt ihr nur unter den `client.loop();` Befehl folgendes ein:

```
1  void loop() {
2      if (!client.connected()) {
3          while (!client.connected()) {
4              client.connect("ESP8266Client");
5              delay(100);
6          }
7      }
8      client.loop();
9      client.publish("/home/data", "Hello World");
10 }
```

Das wars! Der erste Wert ist die Topic, an die GEPUBLISHED wird. Der Zweite Wert ist der Payload bzw. der Wert, den ihr schicken wollt. Wenn ihr hier angekommen seid, dann könnt ihr zum Beispiel schon einen Temperatursensor mit dem D1 Mini basteln, der jede Minute den Wert an den MQTT-Broker sendet.

Subscribe (Abonnieren, empfangen)

Zuerst muss in der Setup-Funktion `client.setCallback(callback);` hinzugefügt werden. Damit wird angegeben, welche Funktion ausgeführt werden soll, wenn eine Nachricht empfangen wird.

```
1  void setup() {
2      Serial.begin(115200);
3      setup_wifi();
4      client.setServer(MQTT_BROKER, 1883);
5      client.setCallback(callback);
6  }
```

Die Funktion heißt in dem Fall `callback` und hat von mir folgenden Inhalt bekommen.

```
1  void callback(char* topic, byte* payload, unsigned int length) {
2      String msg;
3      for (byte i = 0; i < length; i++) {
4          char tmp = char(payload[i]);
5          msg += tmp;
6      }
7      Serial.println(msg);
8  }
```

Alle nötigen Informationen werden in der Funktion übergeben.

Die empfangene Nachricht wird in `payload` übergeben. Der Payload ist ein Array voller Bytes, also Zahlen von 0-255, die erst zu einem Zusammenhängenden String umkonvertiert werden müssen. Dazu wird eine Schleife so oft durchlaufen, wie es Werte in dem Array `payload[]` gibt. Bei jedem Schleifendurchlauf wird der aktuelle Wert des Arrays von einem Byte in einen Char umgewandelt und an die vorher deklarierte Stringvariable msg angehängt. Nachdem die Schleife vollständig durchlaufen ist , steht die Nachricht im Klartext als String in der Variable `msg`.

Zum Schluss müssen wir noch irgendwo angeben, welche Topic eigentlich SUBSCRIBED werden soll.

Dazu fügen wir dem unter dem `client.connect("ESP8266Client");` folgende Zeile ein:

```
client.subscribe("test/test/baum");
```

```
void loop() {
    if (!client.connected()) {
        while (!client.connected()) {
            client.connect("ESP8266Client");
            client.subscribe("test/test/baum");
            delay(100);
        }
    }
    client.loop();
}
```

Tipp

Für den vollständigen Test benötigt man eine Gegenstelle. Sprich: Wenn ich meinen ESP8266 D1 Mini ein Topic SUBSCRIBEN lasse, dann brauche ich auch einen weiteren Client, der auf dieser TOPIC etwas PUBLISHED. Hierzu gibt es praktische kostenfreie Tools.

- MQTT.fx [GUI-Tool unter Windows und Linux]
- mosquitto_client [Kommandozeilen-Tool unter Linux]

Man kann beim „publishen“ auch auf das gleiche Topic „subscriben“.

Ebenso gibt es MQTT Bibliotheken für Python, Java, Java-Script usw.

Codeschnipsel MQTT Publish

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char* SSID = "WIFI_SSID";
const char* PSK = "WIFI_KEY";
const char* MQTT_BROKER = "test.mosquitto.org";

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
    Serial.begin(115200);
    setup_wifi();
    client.setServer(MQTT_BROKER, 1883);
}

void setup_wifi() {
    WiFi.begin(SSID, PSK);

    while (WiFi.status() != WL_CONNECTED) {
        delay(100);
    }

    Serial.println(WiFi.localIP());
}
void loop() {
    if (!client.connected()) {
        while (!client.connected()) {
            client.connect("ESP8266Client");
            delay(100);
        }
    }
    client.loop();
    client.publish("/home/data", "Hello World");
}
```

Codeschnipsel MQTT Subscribe

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char* SSID = "WIFI_SSID";
const char* PSK = "WIFI_KEY";
const char* MQTT_BROKER = "test.mosquitto.org";

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
    Serial.begin(115200);
    setup_wifi();
    client.setServer(MQTT_BROKER, 1883);
    client.setCallback(callback);
}

void setup_wifi() {
    WiFi.begin(SSID, PSK);

    while (WiFi.status() != WL_CONNECTED) {
        delay(100);
    }

    Serial.println(WiFi.localIP());
}
void loop() {
    if (!client.connected()) {
        while (!client.connected()) {
            client.connect("ESP8266Client");
            client.subscribe("test/test/baum");
            delay(100);
        }
    }
    client.loop();
}
void callback(char* topic, byte* payload, unsigned int length) {
    String msg;
    for (byte i = 0; i < length; i++) {
        char tmp = char(payload[i]);
        msg += tmp;
    }
    Serial.println(msg);
}
```