Knowledge Transfer

Primary Menu

# Android MQTT Client

Brijesh Thumar / April 21, 2017 / Connectivity

## What is MQTT?

Message Queue Telemetry Transport(MQTT) is a machine-to-machine (M2M)/Internet of Things connectivity protocol. It was designed as an extremely **lightweight publish-subscribe** based messaging protocol for use on top of the **TCP/IP** protocol. It is designed for connections to remote locations where a **small code footprint** is required or the **network bandwidth is limited**. The publish-subscribe messaging pattern **requires a message broker**. The broker is responsible for distributing messages to interested clients based on the topic of a message.

## Real world applications

MQTT has been used in **sensors** communicating to a broker via satellite link, over occasional dial-up connections with

**healthcare providers**, and in a range of **home automation** and **small device** scenarios. It is also ideal for **mobile applications** because of its small size, low power usage, minimized data packets, and efficient distribution of information to one or many receivers.

## MQTT Broker

The broker is primarily responsible for **receiving all messages, filtering them**, decide who is interested in it and then sending the message to all subscribed clients.It also holds the session of all persisted clients including subscriptions and missed messages.**MQTT Mosquitto** is a message broker that implements the MQTT 3.1/3.1.1.There is a **publically accessible sandbox server** for the Eclipse IoT projects available at **iot.eclipse.org:1883** .

## Publish/Subscribe

Multiple clients connect to a broker and subscribe to topics that they are interested in. Clients also connect to the broker and publish messages to topics.

## Android MQTT client

The **Eclipse Paho** project provides open-source client implementations of MQTT.Paho Android Service is an MQTT client library written in Java for developing applications on Android.The MQTT connection is encapsulated within an Android Service that runs in the background of the Android application, keeping it alive when the Android application is switching between different Activities.The Paho Android Service is an interface to the Paho Java MQTT Client library for the Android Platform.

### Installation

The most convenient way to start a new Android Application is to use Android Studio. Download Paho Android Service and Android MQTT Client library.Go to your `libs` folder inside `app` folder and paste all your `.jar`

To add the Paho Android Service as a dependency to you app add the following parts to your gradle file.

```
dependencies {
    compile files('libs/org.eclipse.paho.android.service-1.1.1.jar')
    compile files('libs/org.eclipse.paho.client.mqttv3-1.1.1.jar')
}
```

## Permission

The Paho Android Service needs the following permissions to work

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

To be able to create a binding to the Paho Android Service, the service needs to be declared in the `AndroidManifest.xml`. Add the following within the `<application>` tag

```
<service android:name="org.eclipse.paho.android.service.MqttService" />
```

## MQTT Connection Option

`MqttAndroidClient` will connect with MQTT 3.1.1 by default. To intentionally connect with MQTT 3.1 `MqttConnectOptions` object can be supplied to the `connect` method

## Clean session

On connection, a client sets the “clean session” flag.If the clean session is **set to false**, then the connection is **treated** as durable. This means that when the client disconnects, **any subscriptions it has will remain** and any subsequent QoS 1 or 2 **messages will be stored** until it connects again in the future. If the clean **session is true**, then all subscriptions will be **removed from the client when it disconnects**.

## Automatic Reconnect

Sets whether the client will automatically attempt to reconnect to the server if the connection is lost.If **set to true**, in the event that the connection is lost, the **client will attempt to reconnect** to the server. It will initially **wait 1 second** before it attempts to reconnect, for every failed to reconnect attempt, the **delay will double until it is at 2** minutes at which point the delay will **stay at 2 minutes**.

## Wills

When a client connects to a broker, it may inform the broker that **it has a will**. This is a message that it **wishes the broker to send when the client disconnects unexpectedly**. The will message has a topic, QoS and retains status just the same as any other message.

```
1 private MqttConnectOptions getMqttConnectionOption() {
2         MqttConnectOptions mqttConnectOptions = new MqttConnectOptions();
3         mqttConnectOptions.setCleanSession(false);
```

```
        mqttConnectOptions.setAutomaticReconnect(true);
        mqttConnectOptions.setWill(Constants.PUBLISH_TOPIC, "I am going offline".getBytes(), 1, t
        //mqttConnectOptions.setUserName("username");
        //mqttConnectOptions.setPassword("password".toCharArray());
        return mqttConnectOptions;
}
```

# Receive MQTT Message

```
...
pahoMqttClient = new PahoMqttClient();
mqttAndroidClient = pahoMqttClient.getMqttClient(
getApplicationContext(), Constants.MQTT_BROKER_URL, Constants.CLIENT_ID);

      mqttAndroidClient.setCallback(new MqttCallbackExtended() {
      @Override
      public void connectComplete(boolean b, String s) {

       }
       @Override
       public void connectionLost(Throwable throwable) {

       }
       @Override
       public void messageArrived(String s, MqttMessage mqttMessage) throws Exception {
              setMessageNotification(s, new String(mqttMessage.getPayload()));
       }
       @Override
       public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {

       }
});
....
```

# Disconnect Options

Messages published to the topic if the connection is available, but if it disconnects and connects again **offline messages send**.Holds the set of options that govern the behavior of Offline (or Disconnected) buffering of messages.

```
private DisconnectedBufferOptions getDisconnectedBufferOptions() {
        DisconnectedBufferOptions disconnectedBufferOptions = new DisconnectedBufferOptions();
        disconnectedBufferOptions.setBufferEnabled(true);
        disconnectedBufferOptions.setBufferSize(100);
        disconnectedBufferOptions.setPersistBuffer(true);
        disconnectedBufferOptions.setDeleteOldestMessages(false);
        return disconnectedBufferOptions;
}
```

# Create MQTT Client

Creates an instance of an Android MQTT client, that will bind to the Paho Android Service. By calling the `connect` method of the `MqttAndroidClient` the client will asynchronously try to connect to the MQTT broker and return a token. That token can be used to register callbacks, to get notified when either the MQTT-connection gets connected or an error occurs

```
public MqttAndroidClient getMqttClient(Context context, String brokerUrl, String clientId) {
        mqttAndroidClient = new MqttAndroidClient(context, brokerUrl, clientId);
        try {
            IMqttToken token = mqttAndroidClient.connect(getMqttConnectionOption());
            token.setActionCallback(new IMqttActionListener() {
                @Override
                public void onSuccess(IMqttToken asyncActionToken) {
                    mqttAndroidClient.setBufferOpts(getDisconnectedBufferOptions());
                    Log.d(TAG, "Success");
                }
```

```
11
12                  @Override
13                  public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
14                      Log.d(TAG, "Failure " + exception.toString());
15                  }
16              });
17          } catch (MqttException e) {
18              e.printStackTrace();
19          }
20          return mqttAndroidClient;
21      }
```

# Topics

Messages in MQTT are **published on topics**. There is **no need to configure a topic**, publishing on it is enough. Topics are treated as a hierarchy, **using a slash (/)** as a separator much in the same way as a f**ilesystem**.

Clients can receive messages by **creating subscriptions**. A subscription may be to an **explicit topic**, in which case only messages to that topic will be received, or it **may include wildcards**. Two wildcards are available, `+` or. `#`

# Quality of Service

MQTT defines **3** levels of Quality of Service (QoS). The QoS defines how hard the broker/client will try to ensure that a message is received. Messages may be sent at any QoS level, and clients may attempt to subscribe to topics at any QoS level.

- 0: The broker/client will deliver the message once, with **no confirmation**.
- 1: The broker/client will deliver the message at least once, with **confirmation required**.

- 2: The broker/client will deliver the message exactly once by using a **4 step handshake**.

## Retained Messages

The broker will **keep the message** even after sending it to all current subscribers. If a **new subscription** is made that matches the topic of the **retained message will be sent to the client**. This is useful as a **"last known good"** mechanism. With a retained message, the client will receive an **instant update**.

## Publish Message

The `MqttAndroidClient` allows messages to be published via its `publish` method.

```
public void publishMessage(@NonNull MqttAndroidClient client,
            @NonNull String msg, int qos, @NonNull String topic)
            throws MqttException, UnsupportedEncodingException {
     byte[] encodedPayload = new byte[0];
     encodedPayload = msg.getBytes("UTF-8");
     MqttMessage message = new MqttMessage(encodedPayload);
     message.setId(5866);
     message.setRetained(true);
     message.setQos(qos);
     client.publish(topic, message);
}
```

## Subscribe

Subscriptions can be created via the `MqttAndroidClient.subscribe` method, which takes the topic and the QOS as parameters and returns a `IMqttToken`.

```
public void subscribe(@NonNull MqttAndroidClient client,
```

```
            @NonNull final String topic, int qos) throws MqttException {
        IMqttToken token = client.subscribe(topic, qos);
        token.setActionCallback(new IMqttActionListener() {

        @Override
        public void onSuccess(IMqttToken iMqttToken) {
                Log.d(TAG, "Subscribe Successfully " + topic);
         }

         @Override
         public void onFailure(IMqttToken iMqttToken, Throwable throwable) {
                 Log.e(TAG, "Subscribe Failed " + topic);
         }
    });
}
```

# Unsubscribe

```
public void unSubscribe(@NonNull MqttAndroidClient client,
            @NonNull final String topic) throws MqttException {

        IMqttToken token = client.unsubscribe(topic);
        token.setActionCallback(new IMqttActionListener() {
            @Override
            public void onSuccess(IMqttToken iMqttToken) {
                Log.d(TAG, "UnSubscribe Successfully " + topic);
            }

            @Override
            public void onFailure(IMqttToken iMqttToken, Throwable throwable) {
                Log.e(TAG, "UnSubscribe Failed " + topic);
            }
        });
    }
```

# Disconnect MQTT Client

```
public void disconnect(@NonNull MqttAndroidClient client)
                       throws MqttException {
        IMqttToken mqttToken = client.disconnect();
        mqttToken.setActionCallback(new IMqttActionListener() {
        @Override
        public void onSuccess(IMqttToken iMqttToken) {
                Log.d(TAG, "Successfully disconnected");
        }
        @Override
        public void onFailure(IMqttToken iMqttToken, Throwable throwable) {
                Log.d(TAG, "Failed to disconnected " + throwable.toString());
        }
    });
 }
```

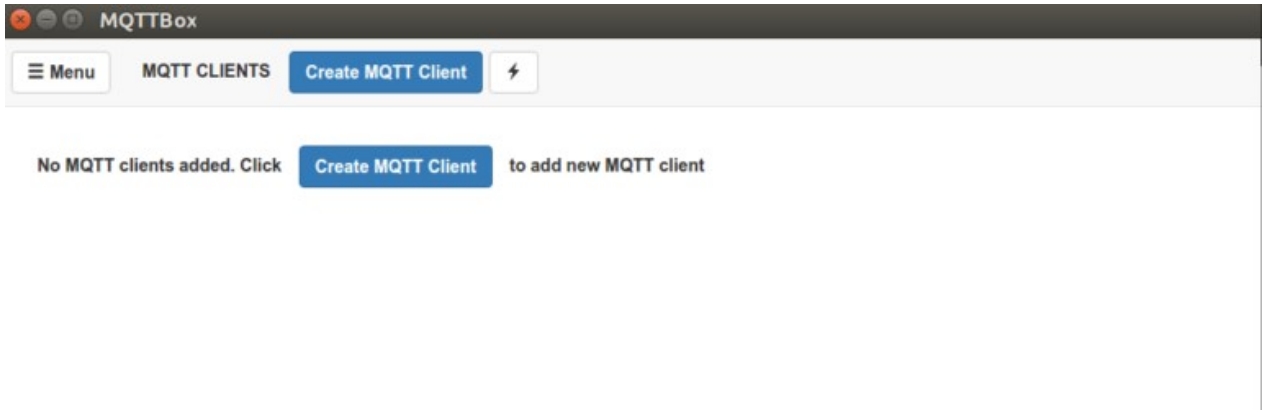
Download this project from GitHub.


# MQTTBox

MQTTBox enables you to create MQTT clients to publish or subscript topics, create MQTT virtual device networks, load test MQTT devices or brokers.

### Config MQTT Client

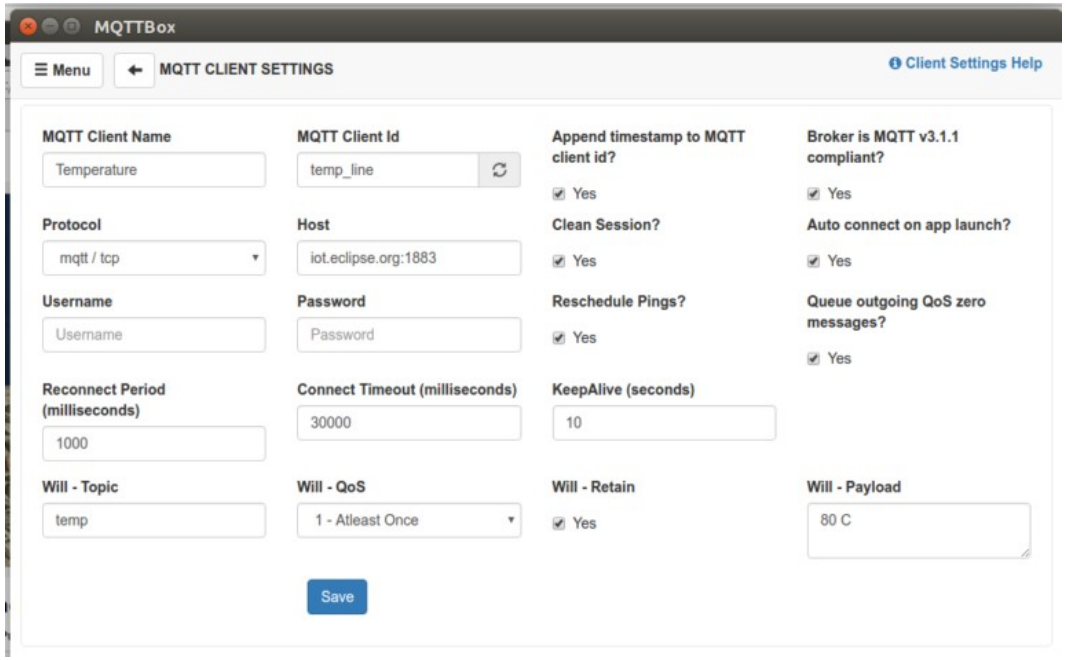
### 1.Create MQTT client



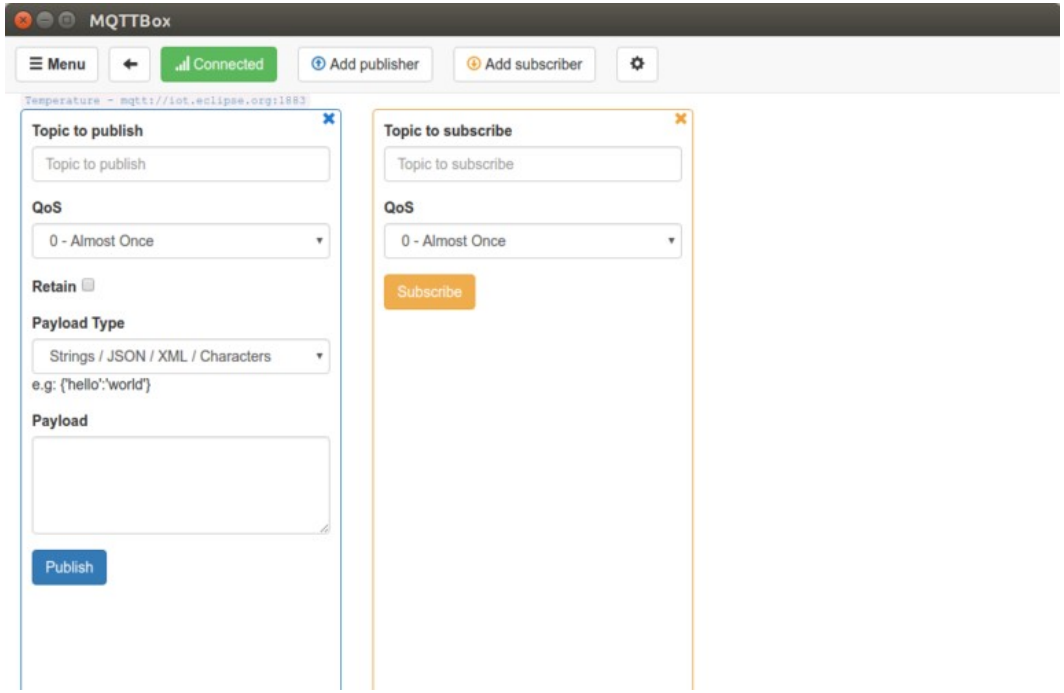

## 2.Config MQTT client

**Host :** iot.eclipse.org:1883

**Protocol :** mqtt/tcp




## 3.Publish/Subscribe




## Related Post

## Android Bluetooth Low Energy Client

## Creating and Monitoring Geofences

**Share this post:** on Twitter on Facebook on Google+

Previous Article
Creating and Monitoring Geofences

Next Article
Bluetooth Low Energy

## 6 Replies to “Android MQTT Client”

**Jazzi** says: July 25, 2017 at 7:47 pm

Really helpful tutorial. Can i use this project with other broker like HiveMq and thingspeak ??

Reply

**brijeshthumar** says: July 25, 2017 at 7:58 pm

yes, you can use any MQTT broker.

Reply

**risa** says: September 30, 2017 at 9:27 am

Hi there, i tried the code, but im not sure why im getting this error

java.lang.NullPointerException: Attempt to invoke virtual method 'org.eclipse.paho.client.mqttv3.IMqttDeliveryToken org.eclipse.paho.android.service.MqttService.publish(java.lang.String, java.lang.String, org.eclipse.paho.client.mqttv3.MqttMessage, java.lang.String, java.lang.String)' on a null object reference

Reply

**Ferraz** says: December 6, 2017 at 10:48 pm

Hi There,

We are working in a project that needs a MQTT broker running on Android Platform. Do you have any idea why we cant find a MQTT broker for Android?

Reply

**Blue** says: December 18, 2017 at 8:27 pm

Hello,
I can't connect to other broker. But MQTTBox can.

Please help me! Thanks

Reply

**NGUYEN VAN QUAN** says:

January 18, 2018 at 10:35 am

Hi, I use your project and it work so good. But How can we check event fail when public message (e.g: network error, etc). I want cancel all error message. Thanks!

Reply

## Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

## Recent Posts

- Calling REST API from a Flutter App
- Guide to Android Architecture Components
- Feeding your own data set into the CNN model in TensorFlow
- Convert a directory of images to TFRecords
- Deep learning model for Car Price prediction using TensorFlow
- Nearby Connections API 2.0
- Room database Migrating

- Rest API Pagination with Paging Library.
- Architecture Components:Paging Library
- Room: Database Relationships
- Image Classify Using TensorFlow Lite
- TensorFlow Lite
- Location Use cases Best Practices
- Understanding Battery Drain when using Location
- How Location API Works in Android

## Categories

- architecture (9)
- Connectivity (3)
- Firebase (8)
- kotlin (1)
- Layout (1)
- Library (1)
- Location (4)
- Machine Learning APIs (3)
- Performance (2)
- TensorFlow (10)
- Uncategorized (15)

## Archives

- March 2018
- February 2018
- January 2018
- December 2017
- November 2017
- October 2017
- September 2017
- August 2017
- July 2017
- June 2017
- May 2017
- April 2017