

QuecPython MQTT User Guide

LTE Standard Module Series

Version: 1.0.0

Date: 2020-11-11 Status:

Preliminary



www.quectel.com

Our aim is to provide customers with timely and comprehensive service. For any assistance, please contact our company headquarters:

Quectel Wireless Solutions Co., Ltd.

Building 5, Shanghai Business Park Phase III (Area B), No.1016 Tianlin Road, Minhang District, Shanghai 200233, China

Tel: +86 21 5108 6236

Email: info@quectel.com

Or our local office. For more information, please visit: <http://www.quectel.com/support/sales.htm>.

For technical support, or to report documentation errors, please visit:

<http://www.quectel.com/support/technical.htm> Or email to support@quectel.com.

General Notes

Quectel offers the information as a service to its customers. The information provided is based upon customers' requirements. Quectel makes every effort to ensure the quality of the information it makes available. Quectel does not make any warranty as to the information contained herein, and does not accept any liability for any injury, loss or damage of any kind incurred by use of or reliance upon the information. All information supplied herein is subject to change without prior notice.

Disclaimer

While Quectel has made efforts to ensure that the functions and features under development are free from errors, it is possible that these functions and features could contain errors, inaccuracies and omissions. Unless otherwise provided by valid agreement, Quectel makes no warranties of any kind, implied or express, with respect to the use of features and functions under development. To the maximum extent permitted by law, Quectel excludes all liability for any loss or damage suffered in connection with the use of the functions and features under development, regardless of whether such loss or damage may have been foreseeable.

Duty of Confidentiality

The Receiving Party shall keep confidential all documentation and information provided by Quectel, except when the specific permission has been granted by Quectel. The Receiving Party shall not access or use Quectel's documentation and information for any purpose except as expressly provided herein. Furthermore, the Receiving Party shall not disclose any of the Quectel's documentation and information to any third party without the prior written consent by Quectel. For any noncompliance to the above requirements, unauthorized use, or other illegal or malicious use of the documentation and information, Quectel will reserve the right to take legal action.

Copyright

The information contained here is proprietary technical information of Quectel wireless solutions co., ltd. Transmitting, reproducing, disseminating and editing this document as well as using the content without permission are forbidden. Offenders will be held liable for payment of damages. All rights are reserved in the event of a patent grant or registration of a utility model or design.

Copyright © Quectel Wireless Solutions Co., Ltd. 2020. All rights reserved.

About the Document

Revision History

Version	Date	Author	Description
-	2020-11-11	Kenney/Rivern	Creation of the document
1.0.0	2020-11-11	Kenney/Rivern	Preliminary

Contents

About the Document.....	2
Contents.....	3
Figure Index.....	4
1 Introduction	5
2 MQTT Overview.....	6
2.1. MQTT Brief Introduction	6
2.1.1. MQTT Design Principle	6
2.1.2. MQTT Business Scenario.....	6
2.2. Publish/Subscribe Model	7
2.3. MQTT Protocol Principle.....	7
2.4. Related Methods in MQTT	8
2.5. Topic.....	8
2.6. Quality of Service.....	9
3 MQTT APIs	9
3.1. API Introduction	9
3.1.1. MQTTClient	9
3.1.2. MQTTClient.set_callback	10
3.1.3. MQTTClient.connect.....	11
3.1.4. MQTTClient.disconnect.....	12
3.1.5. MQTTClient.ping.....	12
3.1.6. MQTTClient.publish	12
3.1.7. MQTTClient.subscribe	13
3.1.8. MQTTClient.check_msg	13
3.1.9. MQTTClient.wait_msg	14
3.1.10. MQTTClient.subscribe.....	14
4 MQTT Development	16
4.1. Building and Testing MQTT Server.....	16
4.1.1. Building MQTT Server	16
4.1.2. Starting MQTT Server	17
4.1.3. Verifying MQTT Server.....	17
4.1.3.1. Installing MQTT Client.....	17
4.1.3.2. Configuring MQTT Client	18
4.1.3.3. Testing MQTT Interaction.....	19
5 Using MQTT For Data Publishing and Subscription.....	21
6 Terms and Abbreviations	24

Figure Index

Figure 1: MQTT Business Scenario	8
Figure 2: MQTT Protocol Principle	9
Figure 3: Help Information of mosquito	17
Figure 3: MQTT Server Is Started	18
Figure 5: Start-up Interface of MQTT.fx	19
Figure 6: Configure MQTT Connection	20
Figure 7: Interaction Between MQTT Server and Client	21
Figure 8: Publish Message	21
Figure 9: Subscribe Topic	21
Figure 10: Connecting to the EVB	22
Figure 11: Tongxinmao Online Client Server	23

1 Introduction

This document takes EC100Y-CN as an example to introduce how to use QuecPython class library API to implement MQTT functions, including MQTT overview, MQTT server building and testing, and data publishing and subscription.

2 MQTT Overview

2.1. MQTT Brief Introduction

MQTT is a broker-based publish/subscribe messaging protocol designed to be open, simple, lightweight and easy to implement. It is designed for connections with remote locations where a "small code footprint" is required or the network bandwidth is limited.

Its simple specification makes MQTT protocol suitable to the IoT scenarios that requires low power consumption and has limited network bandwidth, such as remote sensing data, automobile, smart home, smart city and medical care.

2.1.1. MQTT Design Principle

9 design principles followed by MQTT protocol are:

1. Streamlines features.
2. Adopts publish/subscribe model, making convenience for message delivering.
3. Allows user to dynamically create a topic, lowering operation cost.
4. Reduces the transmission volume to a minimum, improving transmission efficiency.
5. Pays attention to factors such as low bandwidth, high latency, and unstable networks.
6. Supports controlling continuous session.
7. Tolerates various computing capabilities of the client.
8. Provides management of QoS.
9. No mandatory requirements on the type and format of the data to be transmitted, maintaining flexibility.

2.1.2. MQTT Business Scenario

With MQTT protocol, you can connect the device to IoT Cloud service, manage devices and handle data, and finally apply the data to different business scenarios.

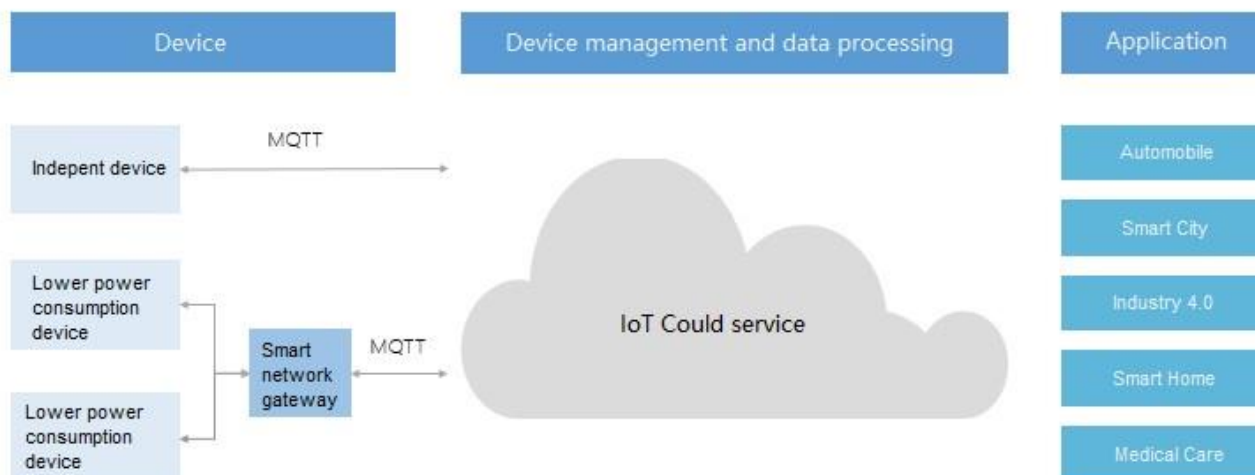


Figure 1: MQTT Business Scenario

2.2. Publish/Subscribe Model

The publish/subscribe model provides an alternative to the traditional client-server architecture. In the client-server model, the client communicates directly with the endpoint. The publish/subscribe model decouples the relationship between the client sending a message (publisher) and the client receiving the message (subscriber), the two do not directly establish a connection. The third component (broker) connects the publisher and the subscriber. The broker filters all incoming messages and distributes them to subscribers correctly.

The publish/subscribe model features:

1. The publisher and subscriber only need to use the same broker without knowing each other.
2. The publisher and subscriber do not need to interact, avoiding locks caused by the publisher waiting for the subscriber's confirmation.
3. The publisher and subscriber do not need to be online at the same time and can freely choose time to publish/receive messages.

2.3. MQTT Protocol Principle



Figure 2: MQTT Protocol Principle

1. To realize the MQTT protocol needs: client and server.
2. There are three identities in the MQTT protocol: publisher, broker, and subscriber. Among them, the message publisher and subscriber are both clients, the message broker is the server, and the message publisher can be both subscribers.
3. The messages transmitted by MQTT are divided into two parts: Topic and Payload:
 - ★ Topic, the type of message, once the subscriber subscribes,, they receive the message content of the topic;
 - ★ Payload, the content of the message, which refers to the content that the subscriber wants to use.

2.4. Related Methods in MQTT

Some methods are defined in MQTT to indicate operations on certain resources. This resource can represent pre-existing data or dynamically generated data, depending on the implementation of the server. Generally, resources refer to files or output on the server:

- ★ Connect, waiting to establish a connection with the server;
- ★ Disconnect, waiting for the MQTT client to finish its work, and disconnect the TCP/IP session with the server;
- ★ Subscribe, waiting to complete the subscription;
- ★ UnSubscribe, waiting for the server to cancel one or more topics subscription of the client;
- ★ Publish, MQTT client sends a message request, and returns to the application thread after sending.

2.5. Topic

MQTT classifies messages by topic. The topic is generally a UTF-8 string. The symbol "/" indicates the hierarchical relationship. The topics can be used directly without creating a new one. The topics can also be filtered by wildcards. "+" filters one level, and "#" generally located at the end of the topic means filtering any level. Examples are as follows:

1. building-b/floor-5: indicates devices from floor 5 of building B.

2. +/floor-5: indicates devices from floor 5 of any building.
3. building-b/#: indicates all devices from building B.

NOTE

MQTT allows subscribing topics with wildcards but not broadcasting.

2.6. Quality of Service

MQTT protocol supports 3 levels of service quality to ensure the reliability of message delivering in different scenarios.

0 At most once.

The sender only sends message one time and not sends repeatedly. It is generally used in the scenarios where unimportant data is transmitted.

1 At least once.

After the message is sent out, the sender waits for the confirmation from the recipient. If the confirmation is not received, the sender sends the message again. This level may result in duplicate messages. It is generally used in log processing scenarios.

2 Only once.

After the message is sent out, the sender waits for the confirmation from the recipient. After receiving the confirmation, the sender deletes the message and notifies the recipient. This level reduces concurrency or increases latency. It is generally used in the scenarios where data loss or repeated messages are unacceptable.

3 MQTT APIs

3.1. API Introduction

3.1.1. MQTTClient

This function constructs the MQTT connection object.

★ Prototype

```
MQTTClient(client_id, server, port=0, user=None, password=None, keepalive=0, ssl=False,
ssl_params={})
```

★ Parameter

client_id:

String type. The client ID is unique, or it may be an encrypted *client_id*, such as Alibaba Cloud.

server:

String type. Server address, which can be IP or domain name.

port:

(Optional) Integer type. The server port, the default is 1883, and the default port of MQTT over SSL/TLS is 8883.

user:

(Optional) String type. The username registered on the server may also be an encrypted user name, such as Alibaba Cloud.

password:

(Optional) String type. The password registered on the server may also be an encrypted password, such as Alibaba Cloud.

keepalive:

(Optional) Integer type. keepalive timeout value of the client. The default is 60 seconds, and the range is 60-120 seconds.

ssl:

(Optional) Boolean. Enable SSL/TLS support or not.

ssl_params:

(Optional) String type. SSL/TLS parameters.

★ Return Value

MQTT object.

3.1.2. MQTTClient.set_callback

This function sets the callback function.

★ Prototype

```
MQTTClient.set_callback(callback)
```

★ Parameter

callback:

Message callback function.

★ Return Value

None.

3.1.3. MQTTClient.connect

This function establishes a connection with the server.

★ Prototype

```
MQTTClient.connect(clean_session=True)
```

★ Parameter

clean_session:

Boolean. Optional parameter, a Boolean value that determines the client type. If it is True, the agent deletes all information about this client when it disconnects. If it is False, the client is a persistent client. When the client disconnects, subscription information and queued messages will be retained. The default is False.

Return Value

None.

3.1.4. MQTTClient.disconnect

This function disconnects from the server.

★ Prototype

```
MQTTClient.disconnect()
```

★ Parameter

None.

★ Return Value

None.

3.1.5. MQTTClient.ping

This function sends a ping packet to the server to detect and maintain connectivity.

★ Prototype

```
MQTTClient.ping()
```

★ Parameter

None.

★ Return Value

None.

3.1.6. MQTTClient.publish

This function publishes messages.

★ Prototype

```
MQTTClient.publish(topic,msg)
```

Parameter*topic:*

String type. Message subject.

msg:

String type. The data to be sent.

★ Return Value

None.

3.1.7. MQTTClient.subscribe

This function subscribes to MQTT topics.

★ Prototype

```
MQTTClient.subscribe(topic,qos)
```

★ Parameter*topic:*

String type. MQTT subject.

msg:

String type. MQTT message service quality, the default is 0, you can choose 0 or 1.

★ Return Value

None.

3.1.8. MQTTClient.check_msg

This function checks whether the server has pending messages.

★ Prototype

```
MQTTClient.check_msg()
```

★ **Parameter**

None.

Return Value

None.

3.1.9. MQTTClient.wait_msg

This function waits for server message response.

★ **Prototype**

```
MQTTClient.wait_msg()
```

★ **Parameter**

None.

★ **Return Value**

None.

3.1.10. MQTTClient.subscribe

This function sets the will to be sent to the server. If the client does not call disconnect() abnormally, it sends a notification to the client.

★ **Prototype**

```
MQTTClient.set_last_will(topic,msg,retain=False,qos=0)
```

★ **Parameter**

topic:

String type. The object of the will.

msg:

String type. The contents of the will.

retain:

Boolean. retain=True broker will always retain the message, the default is False.

msg:

Integer type. Message service quality, the range is 0-2.

Return Value None.

4 MQTT Development

4.1. Building and Testing MQTT Server

Install a MQTT server (broker) on host. This document takes mosquitto (an open source broker) as an example.

4.1.1. Building MQTT Server

1. For Linux, execute the following command to install MQTT server:

```
sudo apt install mosquitto
```

2. For Windows, access <https://mosquitto.org/download/> to download a .exe installation package corresponding to the correct version.

Take the Windows system as an example. After the mosquitto is installed, enter into the installation directory, then start command line and execute the following command to view help information of mosquitto.

```
mosquitto -h
```

```
C:\Program Files\mosquitto>mosquitto -h
mosquitto version 1.6.8

mosquitto is an MQTT v3.1.1 broker.

Usage: mosquitto [-c config_file] [-d] [-h] [-p port]

-c : specify the broker config file.
-d : put the broker into the background after starting.
-h : display this help.
-p : start the broker listening on the specified port.
    Not recommended in conjunction with the -c option.
-v : verbose mode - enable all logging types. This overrides
    any logging options given in the config file.

See http://mosquitto.org/ for more information.
```

Figure 3: Help Information of mosquitto

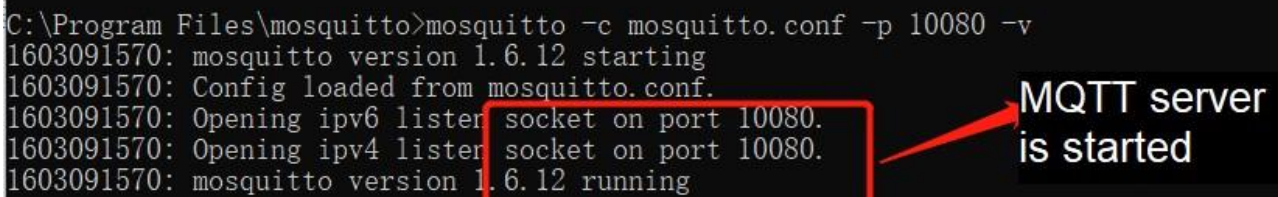
The two key parameters in the above figure -c and -p, -c specifies the configuration of the MQTT service, and -p specifies the service port.

4.1.2. Starting MQTT Server

Before starting the MQTT server, make sure that the network where the server is located can be accessed by the device. Execute the following command in command line to start MQTT:

```
mosquitto -c mosquitto.conf -p 10080 -v
```

In test stage, you can use the default configuration in the installation directory directly for the configuration file; this test uses 10080 as the local port.



```
C:\Program Files\mosquitto>mosquitto -c mosquitto.conf -p 10080 -v
1603091570: mosquitto version 1.6.12 starting
1603091570: Config loaded from mosquitto.conf.
1603091570: Opening ipv6 listener socket on port 10080.
1603091570: Opening ipv4 listener socket on port 10080.
1603091570: mosquitto version 1.6.12 running
```

MQTT server is started

Figure 4: MQTT Server Is Started

4.1.3. Verifying MQTT Server

4.1.3.1. Installing MQTT Client

This document takes MQTT.fx as an example. Please access <http://mqttfx.bceapp.com> to download and install the client. After the MQTT.fx is installed, open it and enter into “Publish” interface.

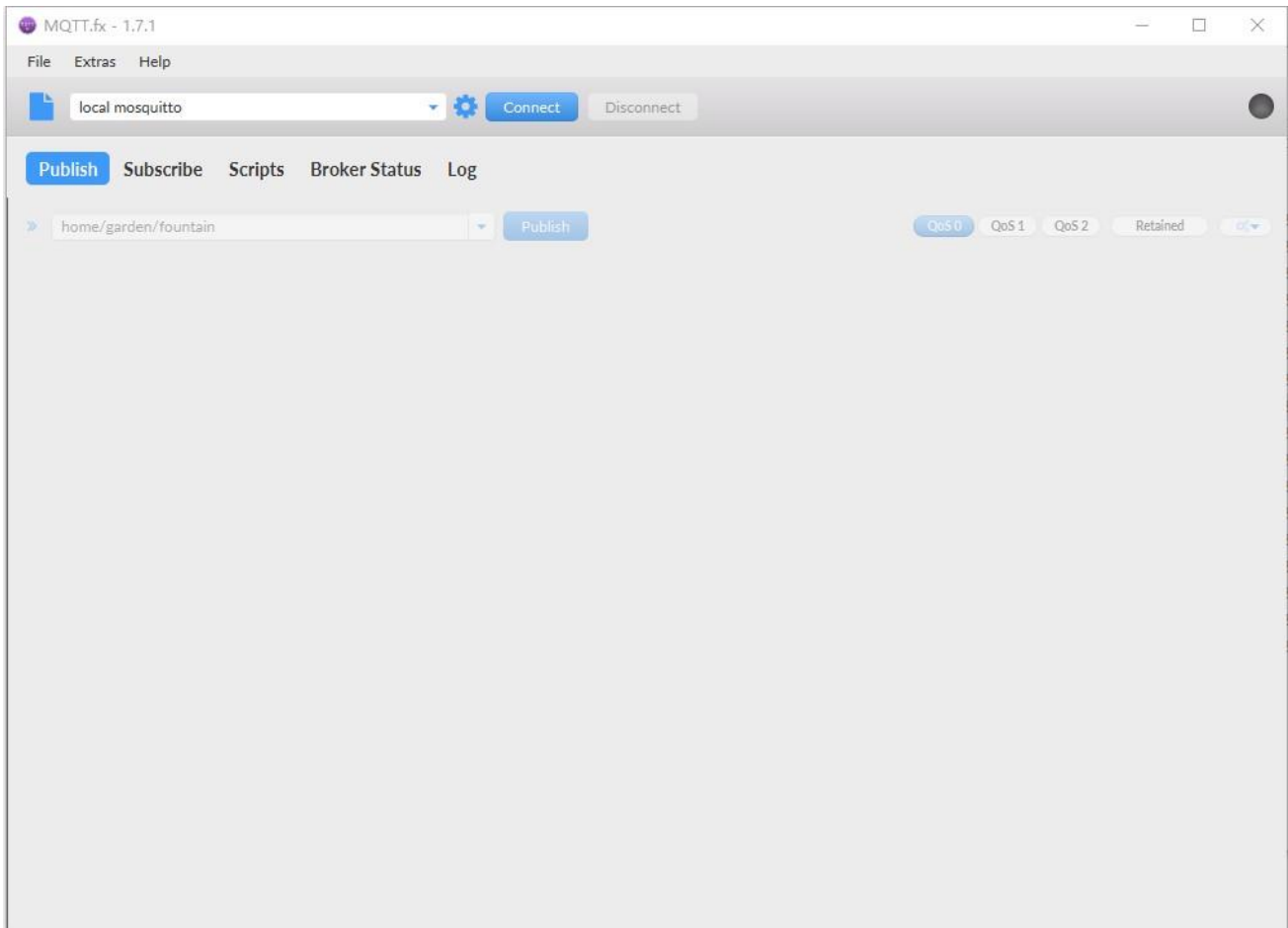


Figure 5: Start-up Interface of MQTT.fx

4.1.3.2. Configuring MQTT Client

Click the button  to configure MQTT connection.

In the “Edit Connection Profiles” page, the configurations of m2m.eclipse.org and mosquitto are displayed by default. You can change any configurations, mainly the “Broker Address” and “Broker Port”. Or you can click the plus symbol at the lower left corner to customize a new connection configuration.

Configure the options “General”, “User Credentials”, “SSL/TLS”, “Proxy” and “LWT” according to the server requirements, including connection timeout, keep-alive interval, MQTT protocol version, username/password and SSL related. This document takes the default configurations as an example.

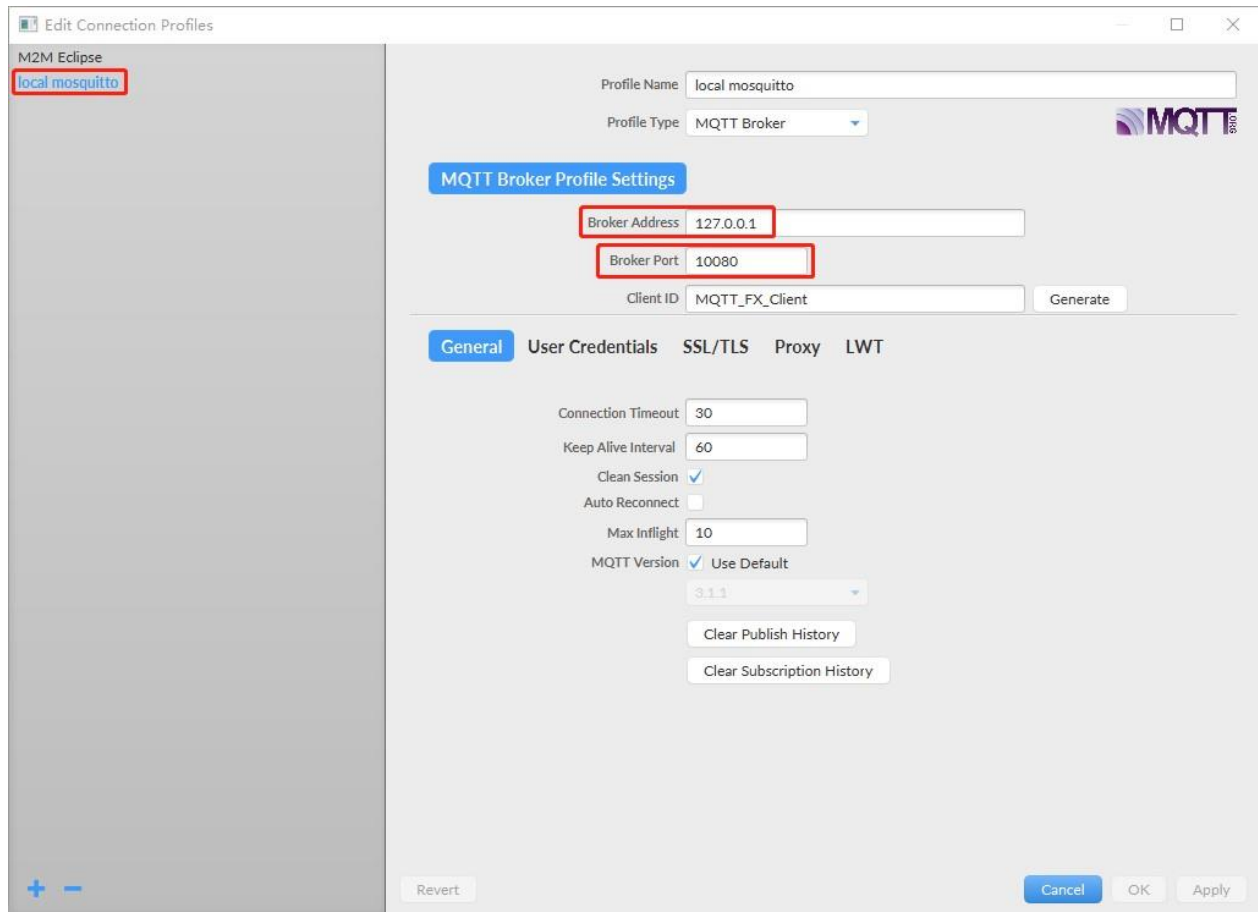


Figure 6: Configure MQTT Connection

4.1.3.3. Testing MQTT Interaction

1. Establish a MQTT connection.

Click **“Connect”** on start-up interface after starting MQTT.fx to establish a MQTT connection. After the connection is successful, please refer to the interaction between MQTT server and client for the mosquitto server prints logs. You can publish message, subscribe topics after the connection is established. The MQTT.fx fills the topics to be subscribed or published in the box by default.

```

1603093267: New client connected from 127.0.0.1 as 5c705e8e04654934bc3dfacbdffbdlf82 (p2, c1, k60).
1603093267: No will message specified.
1603093267: Sending CONNACK to 5c705e8e04654934bc3dfacbdffbdlf82 (0, 0)
1603093276: Received PUBLISH from 5c705e8e04654934bc3dfacbdffbdlf82 (d0, q1, r0, m1, '/wyc/test', ... (8 bytes))
1603093276: Sending PUBACK to 5c705e8e04654934bc3dfacbdffbdlf82 (m1, rc0)
1603093294: Received PUBLISH from 5c705e8e04654934bc3dfacbdffbdlf82 (d0, q1, r0, m2, '/wyc/test', ... (8 bytes))
1603093294: Sending PUBACK to 5c705e8e04654934bc3dfacbdffbdlf82 (m2, rc0)
1603093323: Received SUBSCRIBE from 5c705e8e04654934bc3dfacbdffbdlf82
1603093323: /wyc/test (QoS 0)
1603093323: 5c705e8e04654934bc3dfacbdffbdlf82 0 /wyc/test
1603093323: Sending SUBACK to 5c705e8e04654934bc3dfacbdffbdlf82
1603093326: Received PUBLISH from 5c705e8e04654934bc3dfacbdffbdlf82 (d0, q1, r0, m4, '/wyc/test', ... (8 bytes))
1603093326: Sending PUBACK to 5c705e8e04654934bc3dfacbdffbdlf82 (m4, rc0)
1603093326: Sending PUBLISH to 5c705e8e04654934bc3dfacbdffbdlf82 (d0, q0, r0, m0, '/wyc/test', ... (8 bytes))
1603093386: Received PINGREQ from 5c705e8e04654934bc3dfacbdffbdlf82
1603093386: Sending PINGRESP to 5c705e8e04654934bc3dfacbdffbdlf82

```

Figure 7: Interaction Between MQTT Server and Client

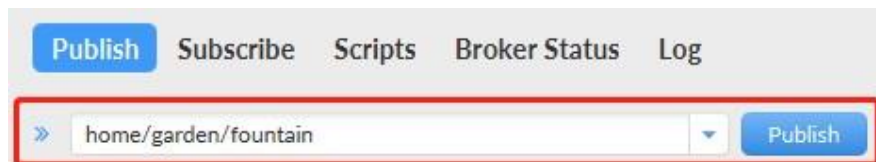


Figure 8: Publish Message



Figure 9: Subscribe Topic

MQTT allows user to create and modify a topic dynamically. Set the topic to be subscribed and to be published to the same one, thus realizing the echo feature. The message published by client to the server is distributed to client immediately.

5 Using MQTT For Data Publishing and Subscription

This test uses the echo function as a test example.

Step1: Connect the EVB to the computer. Please refer to *Quectel_QuecPython_Basic Operation Instructions* for detailed operation methods after connection.

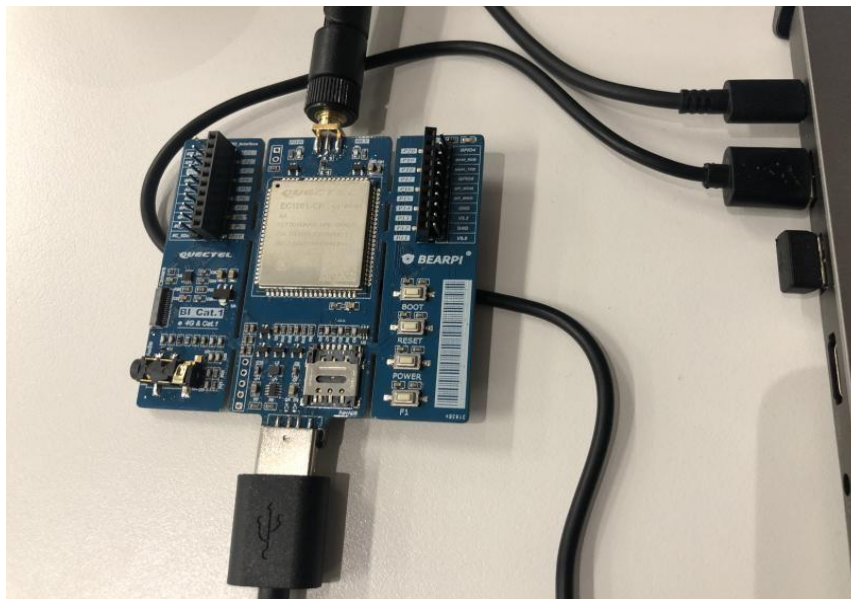


Figure 10: Connecting to the EVB

Step2: Create the *test.py* file, import the umqtt module, we will create the following code by connecting to mq.tongxinmao.com as an example:

```
from umqtt import MQTTClient
state = 0
def sub_cb(topic, msg):
    global state
    print("subscribe recv:")
    print(topic, msg)
    state = 1

#Create an mqtt instance
```

```
c = MQTTClient("umqtt_client", "mq.tongxinmao.com", '18830')
#Set message callback c.set_callback(sub_cb) #Establish
connection c.connect()
#Subscribe to topics
c.subscribe(b"/public/TEST/quecpython") print("Connected to mq.tongxinmao.com,
subscribed to /public/TEST/quecpython topic" )
#Publish message
c.publish(b"/public/TEST/quecpython", b"my name is Kingka!") print("Publish
topic: /public/TEST/quecpython, msg: my name is Quecpython")

while True:
    c.wait_msg() #Block function, listen for messages
    if state == 1:    break

#Close the connection c.disconnect()
```

NOTE

MQTT verification can also be performed by visiting the Tongxinmao online client server:
<http://www.tongxinmao.com/txm/webmqtt.php>.

通信猫 共享MQTT服务器 在线客户端

Connection - Disconnected.(服务器: mq.tongxinmao.com TCP端口18830 WS端口18832 用户/密码:空。简要使用说明: 点击连接-订阅主题-发送消息WIN32客户端下载)

Host主机: mq.tongxinmao.com Port端口: 18832 Username用户名: Password密码: Connect连接

Client ID: txm_1603845670 Path路径: /web Keep-Alive保活: 60 Timeout: 5 TLS: ☐ Clean Session: ☐

Subscribe订阅 (共享服务器TEST帐号订阅主题需以/public/TEST/开头)

Topic主题: /public/TEST/# QoS服务质量: 0最多一次的传输

Subscribe订阅 Unsubscribe取消订阅

Publish发布 (共享服务器TEST帐号发布主题需以/public/TEST/开头)

Topic主题: /public/TEST/webcli QoS: 0最多一次的 Retain: ☐ Publish发布

Message消息: 11 22 33

Figure 11: Tongxinmao Online Client Server

Step3: Upload the *test.py* file to the EC100Y-CN QuecPython EVB. For the detailed upload method, please refer to *Quectel_QuecPython_Basic Operation Instructions*.

Step4: Execute the *test.py* file on the EVB, you can see the execution result, as shown in the figure below:

```
>>> import example
>>> example.exec('test.py')
Connected to mq.tongxinmao.com, subscribed to /public/TEST/quecpython topic
Publish topic: /public/TEST/quecpython, msg: my name is Quecpython
subscribe recv:
b'/public/TEST/quecpython' b'my name is Kingka!'
>>>
```


6 Terms and Abbreviations

Table 1: Terms and Abbreviations

Abbreviation	Description
API	Application Programming Interface
IP	Internet Protocol
MQTT	Message Queuing Telemetry Transport
SSL	Secure Sockets Layer
TLS	Transport Layer Security