mqttools Documentation

Release 0.50.0

Erik Moqvist

Dec 24, 2021

Contents

MQTT tools in Python 3.7 and later.

Both the client and the broker implements MQTT version 5.0 using `asyncio`.

Client features:

- Subscribe to and publish QoS level 0 topics.
- Broker session resume (or clean start support) for less initial communication.
- Topic aliases for smaller publish packets.
- `monitor`, `subscribe` and `publish` command line commands.

Broker features:

- Subscribe to and publish QoS level 0 topics.
- Session resume (or clean start support) for less initial communication. Session state storage in RAM.
- `broker` command line command.

Limitations:

There are lots of limitations in both the client and the broker. Here are a few of them:

- QoS level 1 and 2 messages are not supported. A session state storage is required to do so, both in the client and the broker.
- Authentication is not supported.

MQTT version 5.0 specification: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

Project homepage: https://github.com/eerimoq/mqttools

Documentation: https://mqttools.readthedocs.io

CHAPTER 1

Installation

```
pip install mqttools
```

CHAPTER 2

Examples

There are plenty of examples in the examples folder.

2.1 Command line

2.1.1 Subscribe

Connect to given MQTT broker and subscribe to a topic. All received messages are printed to standard output.

```
$ mqttools subscribe /test/#
Connecting to 'localhost:1883'.
Connected.
Topic:   /test
Message: 11
Topic:   /test/mqttools/foo
Message: bar
```

2.1.2 Publish

Connect to given MQTT broker and publish a message to a topic.

```
$ mqttools publish /test/mqttools/foo bar
Connecting to 'localhost:1883'.

Published 1 message(s) in 0 seconds from 1 concurrent task(s).
```

Publish multiple messages as quickly as possible with `--count` to benchmark the client and the broker.

```
$ mqttools publish --count 100 /test/mqttools/foo
Connecting to 'localhost:1883'.
```

(continues on next page)

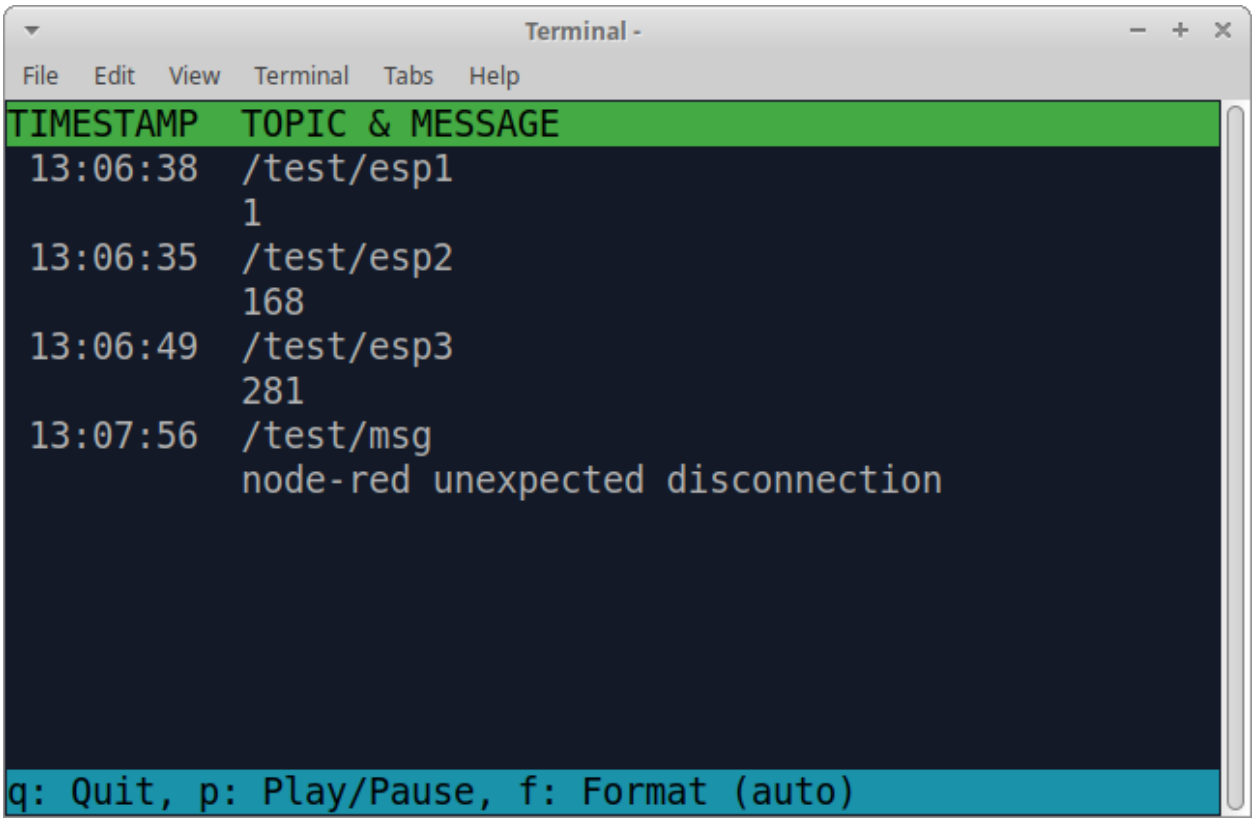
(continued from previous page)

```
Published 100 message(s) in 0.39 seconds from 10 concurrent task(s).
```

2.1.3 Monitor

Connect to given MQTT broker and monitor given topics in a text based user interface.

```
$ mqttools monitor /test/#
```



The menu at the bottom of the monitor shows the available commands.

- Quit: Quit the monitor. Ctrl-C can be used as well.
- Play/Pause: Toggle between playing and paused (or running and freezed).
- Format: Message formatting; *auto*, *binary* or *text*.

2.1.4 Broker

Start a broker to serve clients.

```
$ mqttools broker
```

2.2 Scripting

2.2.1 Subscribe

An example connecting to an MQTT broker, subscribing to the topic `/test/#`, and printing all published messaged.

```
import asyncio
import mqttools

async def subscriber():
    client = mqttools.Client('localhost', 1883)

    await client.start()
    await client.subscribe('/test/#')

    while True:
        message = await client.messages.get()

        if message is None:
            print('Broker connection lost!')
            break

        print(f'Topic:   {message.topic}')
        print(f'Message: {message.message}')

asyncio.run(subscriber())
```

2.2.2 Publish

An example connecting to an MQTT broker and publishing the message `bar` to the topic `/test/mqttools/foo`.

```
import asyncio
import mqttools

async def publisher():
    async with mqttools.Client('localhost', 1883) as client:
        client.publish(mqttools.Message('/test/mqttools/foo', b'bar'))

asyncio.run(publisher())
```

CHAPTER 3

Functions and classes

class mqttools.**Client**(*host*, *port*, *client_id=None*, *will_topic=''*, *will_message=b''*, *will_retain=False*, *will_qos=0*, *keep_alive_s=60*, *response_timeout=5*, *topic_aliases=None*, *topic_alias_maximum=10*, *session_expiry_interval=0*, *subscriptions=None*, *connect_delays=None*, *username=None*, *password=None*, ***kwargs*)

An MQTT version 5.0 client.

host and *port* are the host and port of the broker.

client_id is the client id string. If `None`, a random client id is generated on the form `mqttools-<UUID[0..14]>`.

will_topic, *will_message* and *will_qos* are used to ask the broker to send a will when the session ends.

keep_alive_s is the keep alive time in seconds.

response_timeout is the maximum time to wait for a response from the broker.

topic_aliases is a list of topics that should be published with aliases instead of the topic string.

topic_alias_maximum is the maximum number of topic aliases the client is willing to assign on request from the broker.

session_expiry_interval is the session expiry time in seconds. Give as 0 to remove the session when the connection ends. Give as 0xffffffff to never remove the session (given that the broker supports it).

subscriptions is a list of topics and topic-retain-handling tuples to subscribe to after connected in *start()*.

connect_delays is a list of delays in seconds between the connection attempts in *start()*. Each delay is used once, except the last delay, which is used until successfully connected. If `[]`, only one connection attempt is performed. If `None`, the default delays `[1, 2, 4, 8]` are used.

username and *password* are the credentials. The password must be bytes. By default no credentials are used.

kwargs are passed to `asyncio.open_connection()`.

Create a client with default configuration:

```
>>> client = Client('broker.hivemq.com', 1883)
```

Create a client with all optional arguments given:

```
>>> client = Client('broker.hivemq.com',
                    1883,
                    client_id='my-client',
                    will_topic='/my/last/will',
                    will_message=b'my-last-message',
                    will_qos=0,
                    keep_alive_s=600,
                    response_timeout=30',
                    topic_aliases=['/my/topic']',
                    topic_alias_maximum=100,
                    session_expiry_interval=1800,
                    subscriptions=['a/b', ('test/#', 2)],
                    connect_delays=[1, 2],
                    username='user',
                    password=b'pass',
                    ssl=True)
```

Use an async context manager for automatic start and stop:

```
>>> async with Client('broker.hivemq.com', 1883) as client:
...     client.publish(Message('foo', b'bar'))
```

client_id
The client identifier string.

messages
An `asyncio.Queue` of received messages from the broker. Each message is a *Message*.

```
>>> await client.messages.get()
Message('/my/topic', b'my-message')
```

A `None` message is put in the queue when the broker connection is lost.

```
>>> await client.messages.get()
None
```

publish(*message*)
Publish given message *Message* with QoS 0.

```
>>> client.publish(Message('/my/topic', b'my-message'))
```

on_message(*message*)
Called for each received MQTT message and when the broker connection is lost. Puts the message on the messages queue by default.

start(*resume_session=False*)
Open a TCP connection to the broker and perform the MQTT connect procedure. This method must be called before any *publish()* or *subscribe()* calls. Call *stop()* to close the connection.

If *resume_session* is `True`, the client tries to resume the last session in the broker. A *SessionResumeError* exception is raised if the resume fails, and a new session has been created instead.

The exceptions below are only raised if `connect_delays` is `[]`.

Raises `ConnectionRefusedError` if the TCP connection attempt is refused by the broker.

Raises *`TimeoutError`* if the broker does not acknowledge the connect request.

Raises *`ConnectError`* if the broker does not accept the connect request.

Raises *`SubscribeError`* if the broker does not accept the subscribe request(s).

```
>>> await client.start()
```

Trying to resume a session.

```
>>> try:
...     await client.start(resume_session=True)
...     print('Session resumed.')
... except SessionResumeError:
...     print('Session not resumed. Subscribe to topics.')
```

`stop`()

Try to cleanly disconnect from the broker and then close the TCP connection. Call *`start()`* after *`stop()`* to reconnect to the broker.

```
>>> await client.stop()
```

`subscribe`(*topic*, *retain_handling=0*)

Subscribe to given topic with QoS 0.

retain_handling controls the the retain handling. May be 0, 1 or 2. If 0, all retained messages matching given topic filter are received. If 1, same as 0, but only if the topic filter did not already exist. If 2, no retained messages are received.

Raises *`TimeoutError`* if the broker does not acknowledge the subscribe request.

Raises *`SubscribeError`* if the broker does not accept the subscribe request.

```
>>> await client.subscribe('/my/topic')
>>> await client.messages.get()
Message('/my/topic', b'my-message')
```

`unsubscribe`(*topic*)

Unsubscribe from given topic.

Raises *`TimeoutError`* if the broker does not acknowledge the unsubscribe request.

Raises *`UnsubscribeError`* if the broker does not accept the unsubscribe request.

```
>>> await client.unsubscribe('/my/topic')
```

class mqttools.**Message**(*topic*, *message*, *retain=False*, *response_topic=None*)

A message.

Give *retain* as `True` to make the message retained.

Give *response_topic* as a topic string to publish a response topic.

class mqttools.**Broker**(*addresses*)

A limited MQTT version 5.0 broker.

addresses is a list of `(host, port)` and `(host, port, ssl)` tuples. It may also be the host string or one of the tuples. The broker will listen for clients on all given addresses. `ssl` is an SSL context passed to *asyncio.start_server()* as *ssl*.

Create a broker and serve clients:

```
>>> broker = Broker('localhost')
>>> await broker.serve_forever()
```

serve_forever()
Setup a listener socket and forever serve clients. This coroutine only ends if cancelled by the user.

class mqttools.**BrokerThread**(*addresses*)
The same as *Broker*, but running in a thread.

Create a broker and serve clients for 60 seconds:

```
>>> broker = BrokerThread('broker.hivemq.com')
>>> broker.start()
>>> time.sleep(60)
>>> broker.stop()
```

start()
Start the broker in a thread. This function returns immediately.

stop()
Stop the broker. All clients will be disconnected and the thread will terminate.

class mqttools.**ConnectError**(*reason*)

class mqttools.**SessionResumeError**

class mqttools.**SubscribeError**(*reason*)

class mqttools.**UnsubscribeError**(*reason*)

class mqttools.**TimeoutError**

Index