HBMQTT Documentation

Release 0.6

Nicolas Jouanin

Dec 06, 2018

Contents

HBMQTT is an open source MQTT client and broker implementation.

Built on top of asyncio, Python's standard asynchronous I/O framework, HBMQTT provides a straightforward API based on coroutines, making it easy to write highly concurrent applications.

CHAPTER 1

Features

HBMQTT implements the full set of MQTT 3.1.1 protocol specifications and provides the following features:

- Support QoS 0, QoS 1 and QoS 2 messages flow
- Client auto-reconnection on network lost
- Authentication through password file (more methods can be added through a plugin system)
- Basic `$SYS` topics
- TCP and websocket support
- SSL support over TCP and websocket
- Plugin system

CHAPTER 2

Requirements

HBMQTT is built on Python `asyncio` library which was introduced in Python 3.4. Tests have shown that HBMQTT run best with Python 3.4.3. Python 3.5.0 is also fully supported and recommended. Make sure you use one of these version before installing HBMQTT.

CHAPTER 3

Installation

It is not recommended to install third-party library in Python system packages directory. The preferred way for installing HBMQTT is to create a virtual environment and then install all the dependencies you need. Refer to PEP 405 to learn more.

Once you have a environment setup and ready, HBMQTT can be installed with the following command

```
(venv) $ pip install hbmqtt
```

`pip` will download and install HBMQTT and all its dependencies.

CHAPTER 4

User guide

If you need HBMQTT for running a MQTT client or deploying a MQTT broker, the *Quickstart* describes how to use console scripts provided by HBMQTT.

If you want to develop an application which needs to connect to a MQTT broker, the *MQTTClient API* documentation explains how to use HBMQTT API for connecting, publishing and subscribing with a MQTT broker.

If you want to run you own MQTT broker, th *Broker API reference* reference documentation explains how to embed a MQTT broker inside a Python application.

News and updates are listed in the *Changelog*.

4.1 Quickstart

A quick way for getting started with `HBMQTT` is to use console scripts provided for :

- publishing a message on some topic on an external MQTT broker.
- subscribing some topics and getting published messages.
- running a autonomous MQTT broker

These scripts are installed automatically when installing `HBMQTT` with the following command

```
(venv) $ pip install hbmqtt
```

4.1.1 Publishing messages

`hbmqtt_pub` is a command-line tool which can be used for publishing some messages on a topic. It requires a few arguments like broker URL, topic name, QoS and data to send. Additional options allow more complex use case.

Publishing `` `some_data `` to as `/test` topic on is as simple as :

```
$ hbmqtt_pub --url mqtt://test.mosquitto.org -t /test -m some_data
[2015-11-06 22:21:55,108] :: INFO - hbmqtt_pub/5135-MacBook-Pro.local Connecting to␣
↪broker
[2015-11-06 22:21:55,333] :: INFO - hbmqtt_pub/5135-MacBook-Pro.local Publishing to '/
↪test'
[2015-11-06 22:21:55,336] :: INFO - hbmqtt_pub/5135-MacBook-Pro.local Disconnected␣
↪from broker
```

This will use insecure TCP connection to connect to test.mosquitto.org. `hbmqtt_pub` also allows websockets and secure connection:

```
$ hbmqtt_pub --url ws://test.mosquitto.org:8080 -t /test -m some_data
[2015-11-06 22:22:42,542] :: INFO - hbmqtt_pub/5157-MacBook-Pro.local Connecting to␣
↪broker
[2015-11-06 22:22:42,924] :: INFO - hbmqtt_pub/5157-MacBook-Pro.local Publishing to '/
↪test'
[2015-11-06 22:22:52,926] :: INFO - hbmqtt_pub/5157-MacBook-Pro.local Disconnected␣
↪from broker
```

`hbmqtt_pub` can read from file or stdin and use data read as message payload:

```
$ some_command | hbmqtt_pub --url mqtt://localhost -t /test -l
```

See *hbmqtt_pub* reference documentation for details about available options and settings.

4.1.2 Subscribing a topic

`hbmqtt_sub` is a command-line tool which can be used to subscribe for some pattern(s) on a broker and get date from messages published on topics matching these patterns by other MQTT clients.

Subscribing a `/test/#` topic pattern is done with :

```
$ hbmqtt_sub --url mqtt://localhost -t /test/#
```

This command will run forever and print on the standard output every messages received from the broker. The `-n` option allows to set a maximum number of messages to receive before stopping.

See *hbmqtt_sub* reference documentation for details about available options and settings.

4.1.3 URL Scheme

HBMQTT command line tools use the `--url` to establish a network connection with the broker. The `--url` parameter value must conform to the MQTT URL scheme. The general accepted form is :

```
[mqtt|ws][s]://[username][:password]@host.domain[:port]
```

Here are some examples of URL:

```
mqtt://localhost
mqtt://localhost:1884
mqtt://user:password@localhost
ws://test.mosquitto.org
wss://user:password@localhost
```

4.1.4 Running a broker

`hbmqtt` is a command-line tool for running a MQTT broker:

```
$ hbmqtt
[2015-11-06 22:45:16,470] :: INFO - Listener 'default' bind to 0.0.0.0:1883 (max_
↪connections=-1)
```

See *hbmqtt* reference documentation for details about available options and settings.

4.2 Changelog

4.2.1 0.9.5

- fix more issues
- fix a few issues

4.2.2 0.9.2

- fix a few issues

4.2.3 0.9.1

- See commit log

4.2.4 0.9.0

- fix a serie of issues
- improve plugin performance
- support Python 3.6
- upgrade to `websockets` 3.3.0

4.2.5 0.8.0

- fix a serie of issues

4.2.6 0.7.3

- fix deliver message client method to raise TimeoutError (#40)
- fix topic filter matching in broker (#41)

Version 0.7.2 has been jumped due to troubles with pypi...

4.2.7 0.7.1

- Fix duplicated $SYS topic name .

4.2.8 0.7.0

- Fix a serie of issues reported by Christoph Krey

4.2.9 0.6.3

- Fix issue #22.

4.2.10 0.6.2

- Fix issue #20 (`mqtt` subprotocol was missing).
- Upgrade to `websockets` 3.0.

4.2.11 0.6.1

- Fix issue #19

4.2.12 0.6

- Added compatibility with Python 3.5.
- Rewritten documentation.
- Add command-line tools *hbmqtt*, *hbmqtt_pub* and *hbmqtt_sub*.

4.3 References

Reference documentation for HBMQTT console scripts and programming API.

4.3.1 Console scripts

- *hbmqtt_pub* : MQTT client for publishing messages to a broker
- *hbmqtt_sub* : MQTT client for subscribing to a topics and retrieved published messages
- *hbmqtt* : Autonomous MQTT broker

4.3.2 Programming API

- *MQTTClient API* : MQTT client API reference
- *Broker API reference* : MQTT broker API reference
- *Common API* : Common API

TBD

hbmqtt_pub

`hbmqtt_pub` is a MQTT client that publishes simple messages on a topic from the command line.

Usage

`hbmqtt_pub` usage :

```
hbmqtt_pub --version
hbmqtt_pub (-h | --help)
hbmqtt_pub --url BROKER_URL -t TOPIC (-f FILE | -l | -m MESSAGE | -n | -s) [-c CONFIG_
↪FILE] [-i CLIENT_ID] [-d]
           [-q | --qos QOS] [-d] [-k KEEP_ALIVE] [--clean-session]
           [--ca-file CAFILE] [--ca-path CAPATH] [--ca-data CADATA]
           [ --will-topic WILL_TOPIC [--will-message WILL_MESSAGE] [--will-qos WILL_
↪QOS] [--will-retain] ]
           [--extra-headers HEADER]
```

Note that for simplicity, `hbmqtt_pub` uses mostly the same argument syntax as mosquitto_pub.

Options

--version	HBMQTT version information
-h, --help	Display `hbmqtt_pub` usage help
-c	Set the YAML configuration file to read and pass to the client runtime.
-d	Enable debugging informations.
--ca-file	Define the path to a file containing PEM encoded CA certificates that are trusted. Used to enable SSL communication.
--ca-path	Define the path to a directory containing PEM encoded CA certificates that are trusted. Used to enable SSL communication.
--ca-data	Set the PEM encoded CA certificates that are trusted. Used to enable SSL communication.
--clean-session	If given, set the CONNECT clean session flag to True.
-f	Send the contents of a file as the message. The file is read line by line, and `hbmqtt_pub` will publish a message for each line read.
-i	The id to use for this client. If not given, defaults to `hbmqtt_pub/` appended with the process id and the hostname of the client.
-l	Send messages read from stdin. `hbmqtt_pub` will publish a message for each line read. Blank lines won't be sent.
-k	Set the CONNECT keep alive timeout.
-m	Send a single message from the command line.
-n	Send a null (zero length) message.
-q, --qos	Specify the quality of service to use for the message, from 0, 1 and 2. Defaults to 0.
-s	Send a message read from stdin, sending the entire content as a single message.

-t	The MQTT topic on which to publish the message.
--url	Broker connection URL, conforming to MQTT URL scheme.
--will-topic	The topic on which to send a Will, in the event that the client disconnects unexpectedly.
--will-message	Specify a message that will be stored by the broker and sent out if this client disconnects unexpectedly. This must be used in conjunction with `--will-topic`.
--will-qos	The QoS to use for the Will. Defaults to 0. This must be used in conjunction with `--will-topic`.
--will-retain	If given, if the client disconnects unexpectedly the message sent out will be treated as a retained message. This must be used in conjunction with `--will-topic`.
--extra-headers	Specify a JSON object string with key-value pairs representing additional headers that are transmitted on the initial connection, but only when using a websocket connection

Configuration

If `-c` argument is given, `hbmqtt_pub` will read specific MQTT settings for the given configuration file. This file must be a valid YAML file which may contains the following configuration elements :

- `keep_alive` : Keep-alive timeout sent to the broker. Defaults to `10` seconds.
- `ping_delay` : Auto-ping delay before keep-alive timeout. Defaults to 1. Setting to `0` will disable to 0 and may lead to broker disconnection.
- `default_qos` : Default QoS for messages published. Defaults to 0.
- `default_retain` : Default retain value to messages published. Defaults to `false`.
- `auto_reconnect` : Enable or disable auto-reconnect if connectection with the broker is interrupted. Defaults to `false`.
- `reconnect_retries` : Maximum reconnection retries. Defaults to `2`.
- `reconnect_max_interval` : Maximum interval between 2 connection retry. Defaults to `10`.

Examples

Examples below are adapted from mosquitto_pub documentation.

Publish temperature information to localhost with QoS 1:

```
hbmqtt_pub --url mqtt://localhost -t sensors/temperature -m 32 -q 1
```

Publish timestamp and temperature information to a remote host on a non-standard port and QoS 0:

```
hbmqtt_pub --url mqtt://192.168.1.1:1885 -t sensors/temperature -m "1266193804 32"
```

Publish light switch status. Message is set to retained because there may be a long period of time between light switch events:

```
hbmqtt_pub --url mqtt://localhost -r -t switches/kitchen_lights/status -m "on"
```

Send the contents of a file in two ways:

```
hbmqtt_pub --url mqtt://localhost -t my/topic -f ./data

hbmqtt_pub --url mqtt://localhost -t my/topic -s < ./data
```

Publish temperature information to localhost with QoS 1 over mqtt encapsulated in a websocket connection and additional headers:

```
hbmqtt_pub --url wss://localhost -t sensors/temperature -m 32 -q 1 --extra-headers '{
↪"Authorization": "Bearer <token>"}'
```

hbmqtt_sub

`hbmqtt_sub` is a command line MQTT client that subscribes to some topics and output data received from messages published.

Usage

`hbmqtt_sub` usage :

```
hbmqtt_sub --version
hbmqtt_sub (-h | --help)
hbmqtt_sub --url BROKER_URL -t TOPIC... [-n COUNT] [-c CONFIG_FILE] [-i CLIENT_ID] [-
↪q | --qos QOS] [-d]
           [-k KEEP_ALIVE] [--clean-session] [--ca-file CAFILE] [--ca-path CAPATH] [--
↪ca-data CADATA]
           [ --will-topic WILL_TOPIC [--will-message WILL_MESSAGE] [--will-qos WILL_
↪QOS] [--will-retain] ]
           [--extra-headers HEADER]
```

Note that for simplicity, `hbmqtt_sub` uses mostly the same argument syntax as mosquitto_sub.

Options

--version	HBMQTT version information
-h, --help	Display `hbmqtt_sub` usage help
-c	Set the YAML configuration file to read and pass to the client runtime.
-d	Enable debugging informations.
--ca-file	Define the path to a file containing PEM encoded CA certificates that are trusted. Used to enable SSL communication.
--ca-path	Define the path to a directory containing PEM encoded CA certificates that are trusted. Used to enable SSL communication.
--ca-data	Set the PEM encoded CA certificates that are trusted. Used to enable SSL communication.
--clean-session	If given, set the CONNECT clean session flag to True.
-i	The id to use for this client. If not given, defaults to `hbmqtt_sub/` appended with the process id and the hostname of the client.
-k	Set the CONNECT keep alive timeout.

-n	Number of messages to read before ending. Read forever if not given.
-q, --qos	Specify the quality of service to use for receiving messages. This QoS is sent in the subscribe request.
-t	Topic filters to subcribe.
--url	Broker connection URL, conforming to MQTT URL scheme.
--will-topic	The topic on which to send a Will, in the event that the client disconnects unexpectedly.
--will-message	Specify a message that will be stored by the broker and sent out if this client disconnects unexpectedly. This must be used in conjunction with `--will-topic`.
--will-qos	The QoS to use for the Will. Defaults to 0. This must be used in conjunction with `--will-topic`.
--will-retain	If given, if the client disconnects unexpectedly the message sent out will be treated as a retained message. This must be used in conjunction with `--will-topic`.
--extra-headers	Specify a JSON object string with key-value pairs representing additional headers that are transmitted on the initial connection, but only when using a websocket connection

Configuration

If `-c` argument is given, `hbmqtt_sub` will read specific MQTT settings for the given configuration file. This file must be a valid YAML file which may contains the following configuration elements :

- `keep_alive` : Keep-alive timeout sent to the broker. Defaults to `10` seconds.
- `ping_delay` : Auto-ping delay before keep-alive timeout. Defaults to 1. Setting to `0` will disable to 0 and may lead to broker disconnection.
- `default_qos` : Default QoS for messages published. Defaults to 0.
- `default_retain` : Default retain value to messages published. Defaults to `false`.
- `auto_reconnect` : Enable or disable auto-reconnect if connectection with the broker is interrupted. Defaults to `false`.
- `reconnect_retries` : Maximum reconnection retries. Defaults to `2`.
- `reconnect_max_interval` : Maximum interval between 2 connection retry. Defaults to `10`.

Examples

Examples below are adapted from mosquitto_sub documentation.

Subscribe with QoS 0 to all messages published under $SYS/:

```
hbmqtt_sub --url mqtt://localhost -t '$SYS/#' -q 0
```

Subscribe to 10 messages with QoS 2 from /#:

```
hbmqtt_sub --url mqtt://localhost -t /# -q 2 -n 10
```

Subscribe with QoS 0 to all messages published under $SYS/: over mqtt encapsulated in a websocket connection and additional headers:

```
hbmqtt_sub --url wss://localhost -t '$SYS/#' -q 0 --extra-headers '{"Authorization":
↪"Bearer <token>"}'
```

hbmqtt

`hbmqtt` is a command-line script for running a MQTT 3.1.1 broker.

Usage

`hbmqtt` usage :

```
hbmqtt --version
hbmqtt (-h | --help)
hbmqtt [-c <config_file> ] [-d]
```

Options

--version	HBMQTT version information
-h, --help	Display `hbmqtt_sub` usage help
-c	Set the YAML configuration file to read and pass to the client runtime.

Configuration

If `-c` argument is given, `hbmqtt` will read specific MQTT settings for the given configuration file. This file must be a valid YAML file which may contains the following configuration elements :

- `listeners` : network bindings configuration list
- `timeout-disconnect-delay` : client disconnect timeout after keep-alive timeout
- `auth` : authentication configuration

Without the `-c` argument, the broker will run with the following default configuration:

```
listeners:
    default:
        type: tcp
        bind: 0.0.0.0:1883
sys_interval: 20
auth:
    allow-anonymous: true
plugins:
    - auth_file
    - auth_anonymous
```

Using this configuration, `hbmqtt` will start a broker :

- listening on TCP port 1883 on all network interfaces.
- Publishing `$SYS``_ update messages every ``20` seconds.
- Allowing anonymous login, and password file bases authentication.

Configuration example

```
listeners:
    default:
        max-connections: 50000
        type: tcp
    my-tcp-1:
        bind: 127.0.0.1:1883
        my-tcp-2:
        bind: 1.2.3.4:1883
        max-connections: 1000
    my-tcp-ssl-1:
        bind: 127.0.0.1:8883
        ssl: on
        cafile: /some/cafile
        capath: /some/folder
        capath: certificate data
        certfile: /some/certfile
        keyfile: /some/key
    my-ws-1:
        bind: 0.0.0.0:8080
        type: ws
timeout-disconnect-delay: 2
auth:
    plugins: ['auth.anonymous']
    allow-anonymous: true
    password-file: /some/passwd_file
```

This configuration example shows use case of every parameter.

The `listeners` section define 3 bindings :

- `my-tcp-1` : a unsecured TCP listener on port 1883 allowing `1000` clients connections simultaneously
- `my-tcp-ssl-1` : a secured TCP listener on port 8883 allowing `50000` clients connections simultaneously
- `my-ws-1` : a unsecured websocket listener on port 8080 allowing `50000` clients connections simultaneously

Authentication allows anonymous logins and password file based authentication. Password files are required to be text files containing user name and password in the form of :

```
username:password
```

where `password` should be the encrypted password. Use the `mkpasswd -m sha-512` command to build encoded passphrase. Password file example:

```
# Test user with 'test' password encrypted with sha-512
test:$6$l4zQEHEcowc1Pnv4
↪$HHrh8xnsZoLItQ8BmpFHM4r6q5UqK3DnXp2GaTm5zp5buQ7NheY3Xt9f6godVKbEtA.
↪hOC7IEDwnok3pbAOip.
```

MQTTClient API

The *`MQTTClient`* class implements the client part of MQTT protocol. It can be used to publish and/or subscribe MQTT message on a broker accessible on the network through TCP or websocket protocol, both secured or unsecured.

Usage examples

Subscriber

The example below shows how to write a simple MQTT client which subscribes a topic and prints every messages received from the broker :

```
import logging
import asyncio

from hbmqtt.client import MQTTClient, ClientException
from hbmqtt.mqtt.constants import QOS_1, QOS_2

@asyncio.coroutine
def uptime_coro():
    C = MQTTClient()
    yield from C.connect('mqtt://test.mosquitto.org/')
    # Subscribe to '$SYS/broker/uptime' with QOS=1
    # Subscribe to '$SYS/broker/load/#' with QOS=2
    yield from C.subscribe([
            ('$SYS/broker/uptime', QOS_1),
            ('$SYS/broker/load/#', QOS_2),
         ])
    try:
        for i in range(1, 100):
            message = yield from C.deliver_message()
            packet = message.publish_packet
            print("%d:  %s => %s" % (i, packet.variable_header.topic_name, str(packet.
↪payload.data)))
        yield from C.unsubscribe(['$SYS/broker/uptime', '$SYS/broker/load/#'])
        yield from C.disconnect()
    except ClientException as ce:
        logger.error("Client exception: %s" % ce)

if __name__ == '__main__':
    asyncio.get_event_loop().run_until_complete(uptime_coro())
```

When executed, this script gets the default event loop and asks it to run the `uptime_coro` until it completes. `uptime_coro` starts by initializing a *`MQTTClient`* instance. The coroutine then call *`connect()`* to connect to the broker, here `test.mosquitto.org`. Once connected, the coroutine subscribes to some topics, and then wait for 100 messages. Each message received is simply written to output. Finally, the coroutine unsubscribes from topics and disconnects from the broker.

Publisher

The example below uses the *`MQTTClient`* class to implement a publisher. This test publish 3 messages asynchronously to the broker on a test topic. For the purposes of the test, each message is published with a different Quality Of Service. This example also shows to method for publishing message asynchronously.

```
import logging
import asyncio

from hbmqtt.client import MQTTClient
from hbmqtt.mqtt.constants import QOS_1, QOS_2

```

(continues on next page)

(continued from previous page)

```
@asyncio.coroutine
def test_coro():
    C = MQTTClient()
    yield from C.connect('mqtt://test.mosquitto.org/')
    tasks = [
        asyncio.ensure_future(C.publish('a/b', b'TEST MESSAGE WITH QOS_0')),
        asyncio.ensure_future(C.publish('a/b', b'TEST MESSAGE WITH QOS_1', qos=QOS_
↪1)),
        asyncio.ensure_future(C.publish('a/b', b'TEST MESSAGE WITH QOS_2', qos=QOS_
↪2)),
    ]
    yield from asyncio.wait(tasks)
    logger.info("messages published")
    yield from C.disconnect()


@asyncio.coroutine
def test_coro2():
    try:
        C = MQTTClient()
        ret = yield from C.connect('mqtt://test.mosquitto.org:1883/')
        message = yield from C.publish('a/b', b'TEST MESSAGE WITH QOS_0', qos=QOS_0)
        message = yield from C.publish('a/b', b'TEST MESSAGE WITH QOS_1', qos=QOS_1)
        message = yield from C.publish('a/b', b'TEST MESSAGE WITH QOS_2', qos=QOS_2)
        #print(message)
        logger.info("messages published")
        yield from C.disconnect()
    except ConnectException as ce:
        logger.error("Connection failed: %s" % ce)
        asyncio.get_event_loop().stop()


if __name__ == '__main__':
    formatter = "[%(asctime)s] %(name)s {%(filename)s:%(lineno)d} %(levelname)s -
↪%(message)s"
    logging.basicConfig(level=logging.DEBUG, format=formatter)
    asyncio.get_event_loop().run_until_complete(test_coro())
    asyncio.get_event_loop().run_until_complete(test_coro2())
```

As usual, the script runs the publish code through the async loop. `test_coro()` and `test_coro()` are ran in sequence. Both do the same job. `test_coro()` publish 3 messages in sequence. `test_coro2()` publishes the same message asynchronously. The difference appears the looking at the sequence of MQTT messages sent.

`test_coro()` achieves:

```
hbmqtt/YDYY;NNRpYQSy3?o -out-> PublishPacket(ts=2015-11-11 21:54:48.843901,␣
↪fixed=MQTTFixedHeader(length=28, flags=0x0), variable=PublishVariableHeader(topic=a/
↪b, packet_id=None), payload=PublishPayload(data="b'TEST MESSAGE WITH QOS_0'"))
hbmqtt/YDYY;NNRpYQSy3?o -out-> PublishPacket(ts=2015-11-11 21:54:48.844152,␣
↪fixed=MQTTFixedHeader(length=30, flags=0x2), variable=PublishVariableHeader(topic=a/
↪b, packet_id=1), payload=PublishPayload(data="b'TEST MESSAGE WITH QOS_1'"))
hbmqtt/YDYY;NNRpYQSy3?o <-in-- PubackPacket(ts=2015-11-11 21:54:48.979665,␣
↪fixed=MQTTFixedHeader(length=2, flags=0x0), variable=PacketIdVariableHeader(packet_
↪id=1), payload=None)
hbmqtt/YDYY;NNRpYQSy3?o -out-> PublishPacket(ts=2015-11-11 21:54:48.980886,␣
↪fixed=MQTTFixedHeader(length=30, flags=0x4), variable=PublishVariableHeader(topic=a/
↪b, packet_id=2), payload=PublishPayload(data="b'TEST MESSAGE WITH QOS_2'"))
```

(continues on next page)

(continued from previous page)

```
hbmqtt/YDYY;NNRpYQSy3?o <-in-- PubrecPacket(ts=2015-11-11 21:54:49.029691,␣
↪fixed=MQTTFixedHeader(length=2, flags=0x0), variable=PacketIdVariableHeader(packet_
↪id=2), payload=None)
hbmqtt/YDYY;NNRpYQSy3?o -out-> PubrelPacket(ts=2015-11-11 21:54:49.030823,␣
↪fixed=MQTTFixedHeader(length=2, flags=0x2), variable=PacketIdVariableHeader(packet_
↪id=2), payload=None)
hbmqtt/YDYY;NNRpYQSy3?o <-in-- PubcompPacket(ts=2015-11-11 21:54:49.092514,␣
↪fixed=MQTTFixedHeader(length=2, flags=0x0), variable=PacketIdVariableHeader(packet_
↪id=2), payload=None)fixed=MQTTFixedHeader(length=2, flags=0x0),␣
↪variable=PacketIdVariableHeader(packet_id=2), payload=None)
```

while `test_coro2()` runs:

```
hbmqtt/LYRf52W[56SOjW04 -out-> PublishPacket(ts=2015-11-11 21:54:48.466123,␣
↪fixed=MQTTFixedHeader(length=28, flags=0x0), variable=PublishVariableHeader(topic=a/
↪b, packet_id=None), payload=PublishPayload(data="b'TEST MESSAGE WITH QOS_0'"))
hbmqtt/LYRf52W[56SOjW04 -out-> PublishPacket(ts=2015-11-11 21:54:48.466432,␣
↪fixed=MQTTFixedHeader(length=30, flags=0x2), variable=PublishVariableHeader(topic=a/
↪b, packet_id=1), payload=PublishPayload(data="b'TEST MESSAGE WITH QOS_1'"))
hbmqtt/LYRf52W[56SOjW04 -out-> PublishPacket(ts=2015-11-11 21:54:48.466695,␣
↪fixed=MQTTFixedHeader(length=30, flags=0x4), variable=PublishVariableHeader(topic=a/
↪b, packet_id=2), payload=PublishPayload(data="b'TEST MESSAGE WITH QOS_2'"))
hbmqtt/LYRf52W[56SOjW04 <-in-- PubackPacket(ts=2015-11-11 21:54:48.613062,␣
↪fixed=MQTTFixedHeader(length=2, flags=0x0), variable=PacketIdVariableHeader(packet_
↪id=1), payload=None)
hbmqtt/LYRf52W[56SOjW04 <-in-- PubrecPacket(ts=2015-11-11 21:54:48.661073,␣
↪fixed=MQTTFixedHeader(length=2, flags=0x0), variable=PacketIdVariableHeader(packet_
↪id=2), payload=None)
hbmqtt/LYRf52W[56SOjW04 -out-> PubrelPacket(ts=2015-11-11 21:54:48.661925,␣
↪fixed=MQTTFixedHeader(length=2, flags=0x2), variable=PacketIdVariableHeader(packet_
↪id=2), payload=None)
hbmqtt/LYRf52W[56SOjW04 <-in-- PubcompPacket(ts=2015-11-11 21:54:48.713107,␣
↪fixed=MQTTFixedHeader(length=2, flags=0x0), variable=PacketIdVariableHeader(packet_
↪id=2), payload=None)
```

Both coroutines have the same results except that `test_coro2()` manages messages flow in parallel which may be more efficient.

Reference

MQTTClient API

class hbmqtt.client.**MQTTClient**(*client_id=None*, *config=None*, *loop=None*)

MQTT client implementation.

MQTTClient instances provides API for connecting to a broker and send/receive messages using the MQTT protocol.

Parameters

- **client_id** – MQTT client ID to use when connecting to the broker. If none, it will generated randomly by `hbmqtt.utils.gen_client_id()`
- **config** – Client configuration
- **loop** – asynio loop to use

Returns class instance

connect(*uri=None*, *cleansession=None*, *cafile=None*, *capath=None*, *cadata=None*, *extra_headers={}*)
Connect to a remote broker.

At first, a network connection is established with the server using the given protocol (`mqtt`, `mqtts`, `ws` or `wss`). Once the socket is connected, a CONNECT message is sent with the requested informations.

This method is a *coroutine*.

Parameters

- **uri** – Broker URI connection, conforming to MQTT URI scheme. Uses `uri` config attribute by default.
- **cleansession** – MQTT CONNECT clean session flag
- **cafile** – server certificate authority file (optional, used for secured connection)
- **capath** – server certificate authority path (optional, used for secured connection)
- **cadata** – server certificate authority data (optional, used for secured connection)
- **extra_headers** – a dictionary with additional http headers that should be sent on the initial connection (optional, used only with websocket connections)

Returns CONNACK return code

Raise `hbmqtt.client.ConnectException` if connection fails

disconnect()
Disconnect from the connected broker.

This method sends a DISCONNECT message and closes the network socket.

This method is a *coroutine*.

reconnect(*cleansession=None*)
Reconnect a previously connected broker.

Reconnection tries to establish a network connection and send a CONNECT message. Retries interval and attempts can be controled with the `reconnect_max_interval` and `reconnect_retries` configuration parameters.

This method is a *coroutine*.

Parameters **cleansession** – clean session flag used in MQTT CONNECT messages sent for reconnections.

Returns

CONNACK return code

Raise `hbmqtt.client.ConnectException` if re-connection fails after max retries.

ping()
Ping the broker.

Send a MQTT PINGREQ message for response.

This method is a *coroutine*.

publish(*topic*, *message*, *qos=None*, *retain=None*)
Publish a message to the broker.

Send a MQTT PUBLISH message and wait for acknowledgment depending on Quality Of Service

This method is a *coroutine*.

Parameters

- **topic** – topic name to which message data is published
- **message** – payload message (as bytes) to send.
- **qos** – requested publish quality of service : QOS_0, QOS_1 or QOS_2. Defaults to `default_qos` config parameter or QOS_0.
- **retain** – retain flag. Defaults to `default_retain` config parameter or False.

subscribe (*topics*)

Subscribe to some topics.

Send a MQTT SUBSCRIBE message and wait for broker acknowledgment.

This method is a *coroutine*.

Parameters **topics** – array of topics pattern to subscribe with associated QoS.

Returns SUBACK message return code.

Example of `topics` argument expected structure:

```
[
    ('$SYS/broker/uptime', QOS_1),
    ('$SYS/broker/load/#', QOS_2),
]
```

unsubscribe (*topics*)

Unsubscribe from some topics.

Send a MQTT UNSUBSCRIBE message and wait for broker UNSUBACK message.

This method is a *coroutine*.

Parameters **topics** – array of topics to unsubscribe from.

Example of `topics` argument expected structure:

```
['$SYS/broker/uptime', '$SYS/broker/load/#']
```

deliver_message (*timeout=None*)

Deliver next received message.

Deliver next message received from the broker. If no message is available, this methods waits until next message arrives or `timeout` occurs.

This method is a *coroutine*.

Parameters **timeout** – maximum number of seconds to wait before returning. If timeout is not specified or None, there is no limit to the wait time until next message arrives.

Returns instance of *hbmqtt.session.ApplicationMessage* containing received message information flow.

Raises `asyncio.TimeoutError` if timeout occurs before a message is delivered

MQTTClient configuration

The *MQTTClient* `__init__` method accepts a `config` parameter which allow to setup some behaviour and defaults settings. This argument must be a Python dict object which may contain the following entries:

- `keep_alive`: keep alive (in seconds) to send when connecting to the broker (defaults to `10` seconds). *`MQTTClient`* will *auto-ping* the broker if not message is sent within the keep-alive interval. This avoids disconnection from the broker.
- `ping_delay`: *auto-ping* delay before keep-alive times out (defaults to `1` seconds).
- `default_qos`: Default QoS (`0`) used by *`publish()`* if `qos` argument is not given.
- `default_retain`: Default retain (`False`) used by *`publish()`* if `qos` argument is not given.,
- `auto_reconnect`: enable or disable auto-reconnect feature (defaults to `True`).
- `reconnect_max_interval`: maximum interval (in seconds) to wait before two connection retries (defaults to `10`).
- `reconnect_retries`: maximum number of connect retries (defaults to `2`).

Default QoS and default retain can also be overriden by adding a `topics` with may contain QoS and retain values for specific topics. See the following example:

```
config = {
    'keep_alive': 10,
    'ping_delay': 1,
    'default_qos': 0,
    'default_retain': False,
    'auto_reconnect': True,
    'reconnect_max_interval': 5,
    'reconnect_retries': 10,
    'topics': {
        '/test': { 'qos': 1 },
        '/some_topic': { 'qos': 2, 'retain': True }
    }
}
```

With this setting any message published will set with QOS_0 and retain flag unset except for :

- messages sent to `/test` topic : they will be sent with QOS_1
- messages sent to `/some_topic` topic : they will be sent with QOS_2 and retain flag set

In any case, the `qos` and `retain` argument values passed to method *`publish()`* will override these settings.

Broker API reference

The *`Broker`* class provides a complete MQTT 3.1.1 broker implementation. This class allows Python developers to embed a MQTT broker in their own applications.

Usage example

The following example shows how to start a broker using the default configuration:

```
import logging
import asyncio
import os
from hbmqtt.broker import Broker


@asyncio.coroutine
def broker_coro():
```

(continues on next page)

(continued from previous page)

```
    broker = Broker()
    yield from broker.start()


if __name__ == '__main__':
    formatter = "[%(asctime)s] :: %(levelname)s :: %(name)s :: %(message)s"
    logging.basicConfig(level=logging.INFO, format=formatter)
    asyncio.get_event_loop().run_until_complete(broker_coro())
    asyncio.get_event_loop().run_forever()
```

When executed, this script gets the default event loop and asks it to run the `broker_coro` until it completes. `broker_coro` creates *`Broker`* instance and then *`start()`* the broker for serving. Once completed, the loop is ran forever, making this script never stop ...

Reference

Broker API

class hbmqtt.broker.**Broker**(*config=None*, *loop=None*, *plugin_namespace=None*)

MQTT 3.1.1 compliant broker implementation

Parameters

- **config** – Example Yaml config
- **loop** – asyncio loop to use. Defaults to `asyncio.get_event_loop()` if none is given
- **plugin_namespace** – Plugin namespace to use when loading plugin entry_points. Defaults to `hbmqtt.broker.plugins`

start()

Start the broker to serve with the given configuration

Start method opens network sockets and will start listening for incoming connections.

This method is a *coroutine*.

shutdown()

Stop broker instance.

Closes all connected session, stop listening on network socket and free resources.

Broker configuration

The *`Broker`* `__init__` method accepts a `config` parameter which allow to setup some behaviour and defaults settings. This argument must be a Python dict object. For convinience, it is presented below as a YAML file[1].

```
listeners:
    default:
        max-connections: 50000
        type: tcp
    my-tcp-1:
        bind: 127.0.0.1:1883
    my-tcp-2:
```

(continues on next page)

[1] See PyYAML for loading YAML files as Python dict.

(continued from previous page)

```
        bind: 1.2.3.4:1884
        max-connections: 1000
    my-tcp-ssl-1:
        bind: 127.0.0.1:8885
        ssl: on
        cafile: /some/cafile
        capath: /some/folder
        capath: certificate data
        certfile: /some/certfile
        keyfile: /some/key
    my-ws-1:
        bind: 0.0.0.0:8080
        type: ws
timeout-disconnect-delay: 2
auth:
    plugins: ['auth.anonymous'] #List of plugins to activate for authentication among␣
↪all registered plugins
    allow-anonymous: true / false
    password-file: /some/passwd_file
```

The `listeners` section allows to define network listeners which must be started by the *`Broker`*. Several listeners can be setup. `default` subsection defines common attributes for all listeners. Each listener can have the following settings:

- `bind`: IP address and port binding.
- `max-connections`: Set maximum number of active connection for the listener. `0` means no limit.
- `type`: transport protocol type; can be `tcp` for classic TCP listener or `ws` for MQTT over websocket.
- `ssl` enables (`on`) or disable secured connection over the transport protocol.
- `cafile`, `cadata`, `certfile` and `keyfile` : mandatory parameters for SSL secured connections.

The `auth` section setup authentication behaviour:

- `plugins`: defines the list of activated plugins. Note the plugins must be defined in the `hbmqtt.broker.plugins` entry point.
- `allow-anonymous` : used by the internal `hbmqtt.plugins.authentication.AnonymousAuthPlugin` plugin. This parameter enables (`on`) or disable anonymous connection, ie. connection without username.
- `password-file` : used by the internal `hbmqtt.plugins.authentication.FileAuthPlugin` plugin. This parameter gives to path of the password file to load for authenticating users.

Common API

This document describes `HBMQTT` common API both used by *MQTTClient API* and *Broker API reference*.

Reference

ApplicationMessage

class hbmqtt.session.**ApplicationMessage**(*packet_id*, *topic*, *qos*, *data*, *retain*)

ApplicationMessage and subclasses are used to store published message information flow. These objects can

contain different information depending on the way they were created (incoming or outgoing) and the quality of service used between peers.

build_publish_packet(*dup=False*)

Build `hbmqtt.mqtt.publish.PublishPacket` from attributes

Parameters **dup** – force dup flag

Returns `hbmqtt.mqtt.publish.PublishPacket` built from ApplicationMessage instance attributes

data
Publish message payload data

packet_id
Publish message packet identifier

puback_packet
`hbmqtt.mqtt.puback.PubackPacket` instance corresponding to the PUBACK packet in the messages flow. `None` if QoS != QOS_1 or if the PUBACK packet has not already been received or sent.

pubcomp_packet
`hbmqtt.mqtt.puback.PubrelPacket` instance corresponding to the PUBCOMP packet in the messages flow. `None` if QoS != QOS_2 or if the PUBCOMP packet has not already been received or sent.

publish_packet
`hbmqtt.mqtt.publish.PublishPacket` instance corresponding to the PUBLISH packet in the messages flow. `None` if the PUBLISH packet has not already been received or sent.

pubrec_packet
`hbmqtt.mqtt.puback.PubrecPacket` instance corresponding to the PUBREC packet in the messages flow. `None` if QoS != QOS_2 or if the PUBREC packet has not already been received or sent.

pubrel_packet
`hbmqtt.mqtt.puback.PubrelPacket` instance corresponding to the PUBREL packet in the messages flow. `None` if QoS != QOS_2 or if the PUBREL packet has not already been received or sent.

qos
Publish message Quality of Service

retain
Publish message retain flag

topic
Publish message topic

class hbmqtt.session.**IncomingApplicationMessage**(*packet_id*, *topic*, *qos*, *data*, *retain*)
Bases: *hbmqtt.session.ApplicationMessage*

Incoming *ApplicationMessage*.

class hbmqtt.session.**OutgoingApplicationMessage**(*packet_id*, *topic*, *qos*, *data*, *retain*)
Bases: *hbmqtt.session.ApplicationMessage*

Outgoing *ApplicationMessage*.

4.4 License

```
The MIT License (MIT)

Copyright (c) 2015 Nicolas JOUANIN

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

Python Module Index

h

Index