MQ Light Documentation

Release 1.0.2016120100

IBM

Jun 20, 2017

Contents

MQ Light is designed to allow applications to exchange discrete pieces of information in the form of messages. This might sound a lot like TCP/IP networking, and MQ Light does use TCP/IP under the covers, but MQ Light takes away much of the complexity and provides a higher level set of abstractions to build your applications with.

This python module provides the high-level API by which you can interact with the MQ Light runtime.

See https://developer.ibm.com/messaging/mq-light/ for more details.

CHAPTER 1

MQ Light Client API

class mqlight.**Client**(*service*, *client_id=None*, *security_options=None*, *on_started=None*, *on_state_changed=None*, *on_drain=None*)

The Client class represents an MQLight client instance. Once created, the class will initiate connection to the server.

Constructs and starts a new Client.

Parameters

- **service** (*str, list of str or function*) – takes one of three types to define the address of the service to connect to. As a `str` it is a single URL connection. As a `[str, str, ...]` it is a list of URL connections which are each tried in turn until either a connection is successfully established, or all of the URLs have been tried. As a function, which is invoked each time the Client wants to establish a connection. The function prototype must be `func(callback)` and on completion must perform a call to the `callback` function as `callback(error, services)`. Where `error` indicates a failure in generating the service or None to indicate success. The `services` can either be a `str` or a `[str , str, ...]` containing list URLs to be attempted.
- **client_id** (*str or None*) – An identifier that is associated with this client. If none is supplied then a random name will be generated. The identifier must be unique and should two clients have the same identifier then the server will elect which client will be disconnected.
- **security_options** (*dict*) – A optional set of security options, see below for details
- **on_started** (*function or None*) – A function to be called when the Client as successfully connected and reached the `started` state. This function prototype must be `func(client)` where `client` is this instance.
- **on_state_changed** (*function or None*) – A function to be called when the client connection changes state. This function prototype must be `func(client, state, err)` where `client` is this instance, `state` one of started, starting, stopped, stopping, retrying. `err` optional contains an error report that caused the state changed.

- **on_drain** (*function*) – A function to be called when a backlog of messages to be sent have been cleared. The function prototype must be `func(client)`, where `client` is this instance.

Returns This Client's instance.

Raises

- **TypeError** – if an argument was the incorrect type
- ***InvalidArgumentError*** – if an arguments was invalid.

Security options

•user (str) - the user reference when SASL is enabled

•password (str) - the password for the user when SASL is enabled.

•ssl_trust_certificate (str) - file path to the CA trust certificate

•ssl_client_certificate (str) - file path to the client certificate

•ssl_client_key (str)- file path to the client key

•ssl_client_key_passphrase (str)- the passphrase for client key

•ssl_verify_name (True, False) - True(default) will reject the connection of the supplied server certificate does not match the expected server host

get_id()

Returns The client id

get_service()

Returns The service if connected otherwise `None`

get_state()

Returns The state of the client

States

•started - client is connected to the server and ready to process messages.

•starting - client is attempting to connect to the server following a stop event.

•stopped - client is stopped and not connected to the server. This can occur following a user request or a non-recovery connection error

•stopping - occurs before `stopped` state and is closing any current connections.

•retrying - attempting to connect to the server following a recoverable error. Previous states would be `starting` or `started`

is_stopped()

Returns `True` if the Client is in the stopped or stopping state, otherwise `False`

send(*topic*, *data*, *options=None*, *on_sent=None*)
Sends a message to the MQLight service.

Parameters

- **topic** (*str*) – The topic of the message to be sent to.
- **data** (*str or bytearray.*) – Body of the message. A `str` will be send as Text and `bytearray` will be sent as Binary.

- **options** (*dict*) – A set of attributes for the message to be sent, see details below.
- **on_sent** (*function*) – A function to called when the message has been sent. This function prototype must be `func(client, err, topic, data, options)` where `client` is the instance that has completed the send, `err` contains an error message or `None` if was successfully sent `topic` is the topic that the message was sent to, `data` is the body of the message sent, `options` are the message attributes (see below).

Returns `True` if this message was either sent or is the next to be sent or `False` if the message was queued in user memory, because either there was a backlog of messages, or the client was not in a started state.

Raises

- **TypeError** – if an argument had an incorrect type.
- ***RangeError*** – if an argument was out of range
- ***InvalidArgumentError*** – if an argument was invalid
- ***StoppedError*** – if the client is stopped.

Message attributes

- `qos` specifies the quality of service. This can be 1 for at-least-once, where the client waits to receive confirmation of the server received the message before issuing a `on_sent` callback, or 0 for at-most-once, where there is no confirmation and no callback.
- `ttl` specifies the time-to-live of the message in milli-seconds. This is how long the message will persist if sent to a topic that has a subscription that hasn't expired.

start(*on_started=None*)

Will initiate a request to reconnect to the MQ Light service following a stop request.

Parameters **on_started** (*function or None*) – A function to be called when the Client as successfully connected and reached the `started` state. This function prototype must be `func(client)` where `client` is this instance.

Returns The Client instance.

Raises **TypeError** – if on_started argument is not a function.

state

Returns The state of the client

States

- started - client is connected to the server and ready to process messages.
- starting - client is attempting to connect to the server following a stop event.
- stopped - client is stopped and not connected to the server. This can occur following a user request or a non-recovery connection error
- stopping - occurs before `stopped` state and is closing any current connections.
- retrying - attempting to connect to the server following a recoverable error. Previous states would be `starting` or `started`

stop(*on_stopped=None*)

Initiates a stop request and disconnects the client from the server implicitly closing any subscriptions that the client has open. Once the stop has completed the optional callback is performed.

Parameters **on_stopped** (*function or None*) – function to call when the connection is closed. This function prototype must be `func(client, err)` where `client` is the instance that has stopped and `err` will contain any error report that occurred during the stop request

Returns The Client instance.

Raises

- **TypeError** – if the type of any of the arguments is incorrect.
- ***InvalidArgumentError*** – if any of the arguments are invalid.
- **TypeError** – if the on_stopped argument was not a function

subscribe(*topic_pattern*, *share=None*, *options=None*, *on_subscribed=None*, *on_message=None*)

Initiates a subscription with the server and issue message callbacks each time a message arrives for matches topic pattern.

Parameters

- **topic_pattern** (*str*) – The topic to subscribe to.
- **share** (*str or None*) – The share name of the subscription.
- **options** (*dict or None*) – The subscription attributes , see note below
- **on_subscribed** (*function on None*) – A function to call when the subscription has completed. This function prototype must be *func(client, err, pattern, share)*' where `client` is the instance that has completed the subscription, `err` is `None` if the client subscribed successfully otherwise contains an error message, `pattern` is the subscription pattern and `share` is the share name.
- **on_message** (*function on None*) – function to call when a message is received. This function prototype must be `func(message_type, message, delivery)` where `message_type` is 'message' if a wellformed message has been received or 'malformed' if a malformed message has been received `message` is the message contents and `delivery` is the associate information for the message.

Returns The client instance.

Raises

- **TypeError** – if an argument has an incorrect type
- ***RangeError*** – if an argument is not within certain values.
- ***StoppedError*** – if the client is stopped
- ***InvalidArgumentError*** – if an argument is invalid.

Subscription Attributes

•qos - specifies the quality of service. This can be 0 for `at-most-once` and no confirmation is required and 1 for `at-least-once` where a confirmation is required. See `auto_confirm` attribute

•ttl - specifies the time-to-live of the subscription in milli-seconds. This is how long the subscription will persist before being destroyed.

•credit - specifies the link credit: the number of messages that can be sent without confirmation before the server stops delivering messages from the subscription. The default value is 1024 and a value 0 will block messages being received.

•auto_confirm - a value of `True` means the client will automatically confirm messages as they are received and a value of `False` will require the caller to manaully confirm each message. This is performed by the function call within the delivery object of the message

unsubscribe(*topic_pattern*, *share=None*, *options=None*, *on_unsubscribed=None*)

Initiates the disconnection of an existing subscription and thereby stop the flow of messages.

Parameters

- **topic_pattern** (*str*) – the topic_pattern that was supplied in the previous call to subscribe.
- **share** (*str or None*) – the share that was supplied in the previous call to subscribe.
- **options** (*dict or None*) – Subscription attributes, see note below
- **on_unsubscribed** (*function or None*) – Indicates the unsubscribe request has compeleted. This function prototype must be `func(client, err, pattern, share)` where `client` is the instance that has completed the unsubscription, `err` is `None` if the client unsubscribed successfully otherwise contains an error message, `pattern` is the unsubscription pattern and `share` is the share name.

Returns The instance of the client.

Raises

- **TypeError** – if an argument has an incorrect type.
- ***RangeError*** – if an argument is not within certain values.
- ***StoppedError*** – if the client is stopped.
- ***InvalidArgumentError*** – if an argument has an invalid value.

Subscription attributes

•ttl - a value of 0 will result in the subscription being deleted within the server. A positive value will indicate the time in milliseconds that existing and new message persist before they are removed.

exception mqlight.**InvalidArgumentError**

A MQLight error indicating that a given argument is incorrect and cannot be used. The underlying message will highlight which argument is invalid.

exception mqlight.**RangeError**

A MQLight error indicating that a given argument is not within certain values. The underlying message will highlight which argument is out of range.

exception mqlight.**NetworkError**

A MQLight error indicating that an attempted connection or an existing connection has failed. This will relate to a network issue and the client will treat as recovery and attempt reconnection. The underlying message will detail which server it has issue and the reason.

exception mqlight.**ReplacedError**

A MQLight error indicating that the server has detected two clients with the same client id are connected. This is not supported and this client has been disconnected.

exception mqlight.**SecurityError**

A MQLight error indicating a failure to connect to the server due to a security issue. This may relate to the SASL authentication, or SSL. The underlying message will detail which security issue it is and why has been rejected.

exception mqlight.**StoppedError**

A MQLight error indicating a request such as Send, Subscribe and Unsubscribed has been requested while the client is not in a started state.

exception mqlight.**SubscribedError**

A MQLight error indicating that the Subscription request is a duplicated subscription and is not supported. The underlying message will detail the issue.

exception mqlight.**UnsubscribedError**

A MQLight error indicating that a request to unsubscribed has been rejected as no current subscription can be found. The underlying message will detail the issue.

CHAPTER 2

Samples

To run the samples, navigate to the *mqlight/samples/* folder.

Receiver Sample:

usage: recv.py [-h] [-s SERVICE] [-t TOPIC_PATTERN] [-i CLIENT_ID] [–destination-ttl DESTINATION_TTL] [-n SHARE_NAME] [-f FILE] [-d DELAY] [–verbose] [-c FILE] [–client-certificate FILE] [–client-key FILE] [–client-key-passphrase PASSPHRASE] [–no-verify-name SSL_VERIFY_NAME]

Connect to an MQ Light server and subscribe to the specified topic.

optional arguments:

-h, --help show this help message and exit

-s SERVICE, --service SERVICE service to connect to, for example: amqp://user:password@host:5672 or amqps://host:5671 to use SSL/TLS (default: None)

-t TOPIC_PATTERN, --topic-pattern TOPIC_PATTERN subscribe to receive messages matching TOPIC_PATTERN (default: public)

-i CLIENT_ID, --id CLIENT_ID the ID to use when connecting to MQ Light (default: send_[0-9a-f]{7})

--destination-ttl DESTINATION_TTL set destination time-to-live to DESTINATION_TTL seconds (default: None)

-n SHARE_NAME, --share-name SHARE_NAME optionally, subscribe to a shared destination using SHARE_NAMEas the share name.

-f FILE, --file FILE write the payload of the next message received to FILE (overwriting previous file contents then end. (default is to print messages to stdout)

-d DELAY, --delay DELAY delays the confirmation for DELAY seconds each time a message is received. (default: 0)

--verbose print additional information about each message.

ssl arguments:

-c FILE, --trust-certificate FILE use the certificate contained in FILE (in PEM or DER format) to validate the identify of the server. The connection must be secured with SSL/TLS (e.g. the service URL must start with "amqps://")

--client-certificate FILE use the certificate contained in FILE (in PEM format) to supply the identity of the client. The connection must be secured with SSL/TLS

--client-key FILE use the private key contained in FILE (in PEM format) for encrypting the specified client certificate

--client-key-passphrase PASSPHRASE use PASSPHRASE to access the client private key

--no-verify-name SSL_VERIFY_NAME specify to not additionally check the server's common name in the specified trust certificate matches the actual server's DNS name

Sender Sample:

usage: send.py [-h] [-s SERVICE] [-t TOPIC] [-i CLIENT_ID] [–message-ttl MESSAGE_TTL] [-d DELAY] [-r REPEAT] [–sequence] [-f FILE] [–verbose] [-c FILE] [–client-certificate FILE] [–client-key FILE] [–client-key-passphrase PASSPHRASE] [–no-verify-name SSL_VERIFY_NAME] [MESSAGE [MESSAGE ...]]

Send a message to a MQ Light server.

positional arguments: MESSAGE message to be sent (default: ['Hello world!'])

optional arguments:

-h, --help show this help message and exit

-s SERVICE, --service SERVICE service to connect to, for example: amqp://user:password@host:5672 or amqps://host:5671 to use SSL/TLS (default: None)

-t TOPIC, --topic TOPIC send messages to topic TOPIC (default: public)

-i CLIENT_ID, --id CLIENT_ID the ID to use when connecting to MQ Light (default: send_[0-9a-f]{7})

--message-ttl MESSAGE_TTL set message time-to-live to MESSAGE_TTL seconds (default: None)

-d DELAY, --delay DELAY add NUM seconds delay between each request (default: 0)

-r REPEAT, --repeat REPEAT send messages REPEAT times, if REPEAT <= 0 then repeat forever (default: 1)

--sequence prefix a sequence number to the message payload, ignored for binary messages

-f FILE, --file FILE send FILE as binary data. Cannot be specified at the same time as MESSAGE

--verbose print additional information about each message.

ssl arguments:

-c FILE, --trust-certificate FILE use the certificate contained in FILE (in PEM or DER format) to validate the identify of the server. The connection must be secured with SSL/TLS (e.g. the service URL must start with "amqps://")

--client-certificate FILE use the certificate contained in FILE (in PEM format) to supply the identity of the client. The connection mustbe secured with SSL/TLS

--client-key FILE use the private key contained in FILE (in PEM format) for encrypting the specified client certificate

--client-key-passphrase PASSPHRASE use PASSPHRASE to access the client private key

--no-verify-name SSL_VERIFY_NAME specify to not additionally check the server's common name in the specified trust certificate matches the actual server's DNS name

usage: uiworkout.py [-h] [-s SERVICE] [-v] [-c FILE] [–client-certificate FILE] [–client-key FILE] [–client-key-passphrase PASSPHRASE] [–no-verify-name]

UIWorkout Sample:

Send and receives a number of messages to a MQ Light server.

optional arguments:

-h, --help show this help message and exit

-s SERVICE, --service SERVICE service to connect to, for example: amqp://user:password@host:5672 or amqps://host:5671 to use SSL/TLS (default: amqp://localhost)

-v, --verbose Increase the verbose output of the sample

ssl arguments:

-c FILE, --trust-certificate FILE use the certificate contained in FILE (in PEM or DER format) to validate the identify of the server. The connection must be secured with SSL/TLS (e.g. the service URL must start with "amqps://")

--client-certificate FILE use the certificate contained in FILE (in PEM format) to supply the identity of the client. The connection mustbe secured with SSL/TLS

--client-key FILE use the private key contained in FILE (in PEM format) for encrypting the specified client certificate

--client-key-passphrase PASSPHRASE use PASSPHRASE to access the client private key

--no-verify-name specify to not additionally check the server's common name in the specified trust certificate matches the actual server's DNS name

CHAPTER 3

Feedback

You can help shape the product we release by trying out the code and leaving your feedback.

Reporting bugs

If you think you've found a bug, please leave us feedback. To help us fix the bug a log might be helpful. You can get a log by setting the environment variable `MQLIGHT_PYTHON_LOG` to `debug` and by collecting the output that goes to stderr when you run your application.

CHAPTER 4

Release notes

1.0.2016120100

- AMQPS (TLS) support
- Support for both Python 2 and 3
- Various performance improvements
- Improved compatibility

1.0.2015020201b1

- Initial beta release.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index