STOMPyMQ

Igor Mandrichenko

Feb 17, 2022

CONTENTS:

CHAPTER

ONE

STOMP CLIENT

STOMPClient is used to connect to and communicate with the message broker. Here is how to create a client object and connect it to the Broker:

1.1 Using STOMP Client

1.1.1 Connecting to the Broker

```
import stompy

port = 61613
host = "host.domain.com"
client = stompy.connect((host, port))
```

1.1.2 connect() function arguments

stompy.**connect**(*addr_list*, ***args*)

Creates the client object and connects it to the Broker

Parameters

- **addr_list** – a single broker address as tuple (ip_address, port), or a list of tuples - addresses
- **login** (*str*) – login id to use, default: None
- **passcode** (*str*) – pass code to use, default: None
- **headers** (*dict*) – additional headers for the CONNECT frame, default: none

Returns STOMPlient instance connected to the Broker

Return type *STOMPClient*

1.1.3 Reading messages using iterator

```
import stompy

port = 61613
host = "host.domain.com"
client = stompy.connect((host, port))

client.subscribe("/queue/Q")
client.subscribe("/topic/T")

for frame in client:
    command = frame.Command
    headers = frame.headers()              # copy of the headers array
    source = frame["source"]               # headers can be accessed via mapping␣
↪interface
    option = frame.get("option", "none")
    destination = frame.destination        # convenience, same as frame["destination"]
    body_as_bytes = frame.Body
    body_as_text = frame.text              # decoded with UTF-8
    # ... process frame

# iterator stops when the connection closes
```

1.1.4 Reading messages using recv()

```
frame = client.recv()
while frame is not None:
    # ... process frame
    if ...:
        break
    frame = client.recv()
```

1.1.5 Reading messages using callback

```
def process_message(client, frame):
    # process message received by the client
    if ...:
        return frame     # non-False return will stop the loop()

last_frame = client.loop(callback = process_message)
```

The `loop` method can have zero or more positional arguments. First positional arguments will be the `callback`. The remaining positional arguments will be passed as positional arguments to the `callback` in addition to the `client` and the `frame`. The method accepts 2 optional keyword arguments:

transaction str, transaction ID to associate with all the ACKs and NACKs sent automatically during the loop

timeout numeric, time-out used to receive individual frames. In case of time-out, STOMPTimeout exception will be raised

Additional keyword arguments can be specified. They will be passed to the `callback` as is.

For example:

```
client.loop()                       # loop until disconnection without calling any␣
↪callback

def my_callback_1(client, frame):   # no additional arguments
    ...
    if ...:
        return True
client.loop(my_callback_1)          # loop until my_callback_1 returns True

def my_callback_2(client, frame, param1, param2, param3=None):  # 2 positional and 1␣
↪keyword arguments
    ...
    if ...:
        return True
client.loop(my_callback_2, param1_value, param2_value, param3="hello")

def my_callback_3(client, frame, param1, param2, param3=None):  # 2 positional and 1␣
↪keyword arguments
    ...
    if ...:
        return True
client.loop(my_callback_3, param1_value, param2_value, transaction="txn", timeout=10.0,␣
↪param3="hello")
```

The *loop()* method will run the client in the loop, receiving frames from the broker, calling the `callback`, if present. The *loop()* will return once the callback (if any) returns something which evaluates to True or the connection closes. The *loop()* will return the last value returned by the callback or None if the loop stopped due to the disconnection.

1.1.6 Waiting for a receipt

```
def wait_for_receipt(_, frame, receipt):
    # process message received by the client
    return frame is not None and frame.Command == "RECEIPT" and frame["receipt-id"] ==␣
↪receipt

closed = client.loop(wait_for_receipt, receipt="the-receipt") == None
```

1.1.7 Sending ACKs/NACKs

```
client = stompy.connect((host, port))

client.subscribe("/queue/Q", send_acks = False)         # disable auto-sending ACKs

for frame in client:
    if frame.Command == "MESSAGE" and "ack" in frame:
        if ...:
            client.ack(frame["ack"])
```

(continues on next page)

(continued from previous page)

```
        else:
            client.nack(frame["ack"])
```

1.1.8 Sending messages and other frames

```
client = stompy.connect((host, port))
client.send("SEND",
        destination="/queue/Q",
        body="Hello there",                         # can by bytes or str
        source=str(os.getpid())                     # custom header
)
client.message("/queue/Q", "Hello there", source=str(os.getpid()))       # same as above
```

1.1.9 Sending messages and waiting for receipt

```
client = stompy.connect((host, port))
my_receipt = client.message("/queue/Q", "Hello there", receipt=True)       # will␣
↪generate and return receipt-id

def wait_for_receipt(_, frame, receipt):
    # process message received by the client
    return frame is not None and frame.Command == "RECEIPT" and frame["receipt-id"] ==␣
↪receipt

if client.loop(wait_for_receipt, receipt=my_receipt):
    # receipt received
else:
    # connection closed
```

1.1.10 Transactions

```
client = stompy.connect((host, port))
transaction = client.transaction()
trnsaction.message("/queue/Q", "Message part #1")
trnsaction.message("/queue/Q", "Message part #2")
receipt = transaction.commit(receipt=True)

# wait for receipt
if client.loop(wait_for_receipt, receipt=receipt):
    # receipt received
else:
    # connection closed
```

1.2 STOMPClient object methods

class stompy.STOMPClient

STOMPClient constructor does not have any arguments.

connect(*self*, *addr_list*, *login=None*, *passcode=None*, *headers={}*, ***kv_headers*)

Connects to a broker. On successfull connection, sets the following attributes:

client.BrokerAddress - tuple (ip_address, port) - actual address of the broker the connection was established to clint.Connected = True

Parameters

- **addr_list** – a single broker address as tuple (ip_address, port), or a list of tuples - addresses
- **login** (*str*) – login id to use, default: None
- **passcode** (*str*) – pass code to use, default: None
- **headers** (*dict*) – additional headers for the CONNECT frame, default: none
- **kv_headers** – additional headers for the CONNECT frame

Returns CONNECTED frame returned by the broker

disconnect()

Send DISCONNECT frame, wait for receipt and close the connection.

__init__()

STOMPClient constructor does not have any arguments.

__iter__()

The client can be used as an iterator, returning next received frame on every iteration. The iteration stops when the connection closes:

```
client = STOMPClient()
client.connect(...)
for frame in client:
    ...
# connection closed
```

ack(*ack_id*, *transaction=None*)

Send ACK frame

Parameters

- **ack_id** (*str*) – NACK id to send
- **transaction** (*str or None*) – transaction id to associate the ACK with, default: None

loop(**params*, *transaction=None*, *timeout=None*, ***callback_args*)

The method can have zero or more positional arguments. First positional arguments will be the `callback`. The remaining positional arguments will be passed as positional arguments to the `callback` in addition to client and the frame. Keyword arguments will be passed to the `callback` except `transaction` and `timeout`.

The method will run the client in the loop, receiving frames from the broker, calling the `callback`, if present. The loop() will return once the callback (if any) returns something which evaluates to True or the connection closes. The loop() will return the last value returned by the callback or None if the loop stopped due to the disconnection.

Parameters

- `timeout` (*numeric*) – read time-out in seconds, or None
- `transaction` (*str*) – transaction id to associate ACKs and NACKs sent during the loop, or None

Returns The value returned by the last call to the `callback`. If the loop stopped due to disconnecrtion, returns None

`message`(*destination*, *body=b''*, *id=None*, *headers={}*, *receipt=False*, *transaction=None*, ***kv_headers*)
Conventience method to send a message. Uses send().

Parameters

- `destination` (*str*) – destination to send the message to
- `body` (*bytes*) – message body, default - empty
- `id` (*str or None*) – add message-id header, if not None
- `headers` (*dict*) – headers to add to the message, default - empty
- `receipt` (*boolean or str*) – if True or non-empty string, the frame will include "receipt" header. If `receipt` is a str, it will be used as is. If `receipt` is True, the client will generate new receipt id. If `receipt` is False, do not require a receipt.
- `transaction` (*str*) – transaction id to associate the frame with, or None

Returns receipt (str) if the receipt was requested (`receipt` was not False), otherwise None

`nack`(*ack_id*, *transaction=None*)
Send NACK frame

Parameters

- `ack_id` (*str*) – NACK id to send
- `transaction` (*str or None*) – transaction id to associate the NACK with, default: None

`recv`(**params*, ***args*)
Receive next frame. If the next frame is RECEIPT, notify those who are waiting for it and keep receiving. Return None if the connection closed. Raise STOMPError on ERROR.

Parameters `transaction` (*str or None*) – transaction to associate the automatically sent ACK, or None

Returns frame received or None, if the connection was closed

Return type *STOMPFrame* or None

`send`(*command*, *headers={}*, *body=b''*, *transaction=None*, *receipt=False*, ***kv_headers*)
Send the frame. If a receipt was requested, then the frame sent by the client will incude "receipt" header and the method will return the receipt-id:

Parameters

- `command` (*str*) – frame command
- `headers` (*dict*) – frame headers, default - { }
- `body` (*bytes*) – frame body, default - empty body
- `receipt` (*str or boolean*) – if True or non-empty string, the frame will include "receipt" header. If receipt is a str, it will be used as is. If receipt=True, the client will generate new receipt id. If receipt=False, do not require a receipt.

- **`kv_headers`** – additional headers to add to the frame

Returns receipt (str) if the receipt was requested (`receipt` was not False), otherwise None

`subscribe`(*dest*, *ack_mode='auto'*, *send_acks=True*)
Subscribe to messages sent to the specified destination

Parameters

- **`dest`** (*`str`*) – destination
- **`ack_mode`** (*`str`*) – can be either "auto" (default), "client" or "client-individual"
- **`send_acks`** (*`boolean`*) – whether the client should automatically send ACKs received on this scubscription

Returns subscription id

Return type str

`transaction`(*txn_id=None*)
Creates and begins new transaction

Parameters **`txn_id`** (*`str or None`*) – transaction ID or None (default), in which case a new transaction ID will be generated

`unsubscribe`(*sub_id*)
Remove subscription

Parameters **`sub_id`** (*`str`*) – subscription id

1.3 STOMPTransaction object

`STOMPClient.transaction()` method returns STOMPTransaction object, which has the following methods:

`class stompy.client.`**`STOMPTransaction`**(*client*, *txn_id*)

`abort`(*receipt=None*)
Aborts the transaction.

Parameters **`receipt`** (*`str or boolean`*) – if True or non-empty string, the frame will include "receipt" header. If receipt is a str, it will be used as is. If receipt=True, the client will generate new receipt id. If receipt=False, do not require a receipt.

Returns receipt (str) if the receipt was requested (`receipt` was not False), otherwise None

`ack`(*ack_id*, *transaction=None*)
Sends ACK associated with the transaction

Parameters **`string`** – ack id

`commit`(*receipt=None*)
Commits the transaction.

Parameters **`receipt`** (*`str or boolean`*) – if True or non-empty string, the frame will include "receipt" header. If receipt is a str, it will be used as is. If receipt=True, the client will generate new receipt id. If receipt=False, do not require a receipt.

Returns receipt (str) the receipt was requested (`receipt` was not False), otherwise None

`message`(**params*, ***args*)
Sends MESSAGE frame and associates it with the transaction. The method has same arguments as the STOMPClient.message() method.

`nack`(*ack_id*)
Sends NACK associated with the transaction

Parameters **`string`** – ack id

`recv`(*timeout=None*)
Receives next frame from the Broker. If the subscription allows sendig ACKs, the ACK will be associated with the transaction.

`send`(*command*, ***args*)
Sends a STOMP frame to the broker, associating it with the transaction.

Parameters **`command`** (*`str`*) – frame command

Other arguments are the same as for the STOMPClient.send()

1.4 STOMPFrame object

STOMPFrame object represents a STOMP frame received from the Broker

`class stompy.frame.STOMPFrame`(*command=None*, *body=b''*, *headers=None*, ***headers_kv*)
Initializes STOMP Frame object

Parameters

- **`command`** (*`str`*) – frame command
- **`body`** (*`str, bytes`*) – message body
- **`headers`** (*`dict`*) – dictionary with frame headers
- **`headers_kv`** (*`keyword`*) – keyword arguments will be added to the headers

`property destination`
Convenience accessor for the frame destination

`get`(*name*, *default=None*)
Part of mapping interface to the frame headers:

value = frame.get("header-name", default)

`property headers`
Convenience accessor, returns copy of the frame headers dictionary

`property json`
Convenience accessor to interpret the frame body as a JSON object

`property text`
Convenience accessor, converting the frame body to text. Uses the encoding from the content-type header or UTF-8

CHAPTER

TWO

MESSAGE BROKER

CHAPTER THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX