Introduction to RabbitMQ using AMQPStorm All Versions

Examples

How to consume messages from RabbitMQ

Start with importing the library.

```
from amqpstorm import Connection
```

When consuming messages, we first need to define a function to handle the incoming messages. This can be any callable function, and has to take a message object, or a message tuple (depending on the to_tuple parameter defined in start_consuming).

Besides processing the data from the incoming message, we will also have to Acknowledge or Reject the message. This is important, as we need to let RabbitMQ know that we properly received and processed the message.

```
def on_message(message):
    """This function is called on message received.

    :param message: Delivered message.
    :return:
    """
    print("Message:", message.body)

    # Acknowledge that we handled the message without any issues.
    message.ack()

    # Reject the message.
    # message.reject()

    # Reject the message, and put it back in the queue.
    # message.reject(requeue=True)
```

Next we need to set up the connection to the RabbitMQ server.

```
connection = Connection('127.0.0.1', 'guest', 'guest')
```

After that we need to set up a channel. Each connection can have multiple channels, and in general when performing multi-threaded tasks, it's recommended (but not required) to have one per thread.

```
channel = connection.channel()
```

Once we have our channel set up, we need to let RabbitMQ know that we want to start consuming messages. In this case we will use our previously defined on_message function to handle all our consumed messages.

The queue we will be listening to on the RabbitMQ server is going to be simple_queue , and we are also telling RabbitMQ that we will be acknowledging all incoming messages once we are done with them.

```
channel.basic.consume(callback=on_message, queue='simple_queue', no_ack=False)
```

Finally we need to start the IO loop to start processing messages delivered by the RabbitMQ server.

```
channel.start_consuming(to_tuple=False)
```

How to create a delayed queue in RabbitMQ

First we need to set up two basic channels, one for the main queue, and one for the delay queue. In my example at the end, I include a couple of additional flags that are not required, but makes the code more reliable; such as confirm delivery , delivery_mode and durable . You can find more information on these in the RabbitMQ manual .

After we have set up the channels we add a binding to the main channel that we can use to send messages from the delay channel to our main queue.

```
channel.queue.bind(exchange='amq.direct', routing_key='hello', queue='hello')
```

Next we need to configure our delay channel to forward messages to the main queue once they have expired.

```
delay_channel.queue.declare(queue='hello_delay', durable=True, arguments={
```

```
    'x-message-ttl': 5000,
    'x-dead-letter-exchange': 'amq.direct',
    'x-dead-letter-routing-key': 'hello'
})
```

- x-message-ttl *(Message - Time To Live)*

  This is normally used to automatically remove old messages in the queue after a specific duration, but by adding two optional arguments we can change this behaviour, and instead have this parameter determine in milliseconds how long messages will stay in the delay queue.

- x-dead-letter-routing-key

  This variable allows us to transfer the message to a different queue once they have expired, instead of the default behaviour of removing it completely.

- x-dead-letter-exchange

  This variable determines which Exchange used to transfer the message from hello_delay to hello queue.

Publishing to the delay queue

When we are done setting up all the basic Pika parameters you simply send a message to the delay queue using basic publish.

```
delay_channel.basic.publish(exchange='',
                            routing_key='hello_delay',
                            body='test',
                            properties={'delivery_mod': 2})
```

Once you have executed the script you should see the following queues created in your RabbitMQ management module.

Overview					Messages			Message rates		
Name	**Exclusive**	Parameters	**Policy**	**Status**	**Ready**	**Unacked**	**Total**	**incoming**	**deliver / get**	**ack**
hello		D		Idle	1	0	1			
hello_delay		TTL DLX DLK D		Idle	0	0	0	0.00/s		

Example.

```
from amqpstorm import Connection

connection = Connection('127.0.0.1', 'guest', 'guest')

# Create normal 'Hello World' type channel.
channel = connection.channel()
channel.confirm_deliveries()
channel.queue.declare(queue='hello', durable=True)

# We need to bind this channel to an exchange, that will be used to transfer
# messages from our delay queue.
channel.queue.bind(exchange='amq.direct', routing_key='hello', queue='hello')

# Create our delay channel.
delay_channel = connection.channel()
delay_channel.confirm_deliveries()

# This is where we declare the delay, and routing for our delay channel.
delay_channel.queue.declare(queue='hello_delay', durable=True, arguments={
    'x-message-ttl': 5000, # Delay until the message is transferred in milliseconds.
    'x-dead-letter-exchange': 'amq.direct', # Exchange used to transfer the message from A to B.
    'x-dead-letter-routing-key': 'hello' # Name of the queue we want the message transferred to.
})

delay_channel.basic.publish(exchange='',
                            routing_key='hello_delay',
                            body='test',
                            properties={'delivery_mode': 2})

print("[x] Sent")
```

How to publish messages to RabbitMQ

Start with importing the library.

```
from amqpstorm import Connection
from amqpstorm import Message
```

Next we need to open a connection to the RabbitMQ server.

```
connection = Connection('127.0.0.1', 'guest', 'guest')
```

After that we need to set up a channel. Each connection can have multiple channels, and in general when performing multi-threaded tasks, it's recommended (but not required) to have one per thread.

```
channel = connection.channel()
```

Once we have our channel set up, we can start to prepare our message.

```
# Message Properties.
properties = {
    'content_type': 'text/plain',
    'headers': {'key': 'value'}
}

# Create the message.
message = Message.create(channel=channel, body='Hello World!', properties=properties)
```

Now we can publish the message by simply calling `publish` and providing a `routing_key` . In this case we are going to send the message to a queue called `simple_queue` .

```
message.publish(routing_key='simple_queue')
```

Syntax

Parameters

Remarks

The latest version of AMQPStorm is available at pypi or you can install it using pip

```
pip install amqpstorm
```