

API Documentation

API Documentation

May 1, 2010

Contents

Contents	1
1 Package amqplib	3
1.1 Modules	3
1.2 Variables	3
2 Package amqplib.client_0_8	4
2.1 Modules	4
2.2 Class Message	4
2.2.1 Methods	6
2.2.2 Properties	7
2.2.3 Class Variables	7
2.3 Class Connection	7
2.3.1 Methods	8
2.3.2 Properties	10
2.4 Class AMQPException	10
2.4.1 Methods	10
2.4.2 Properties	11
2.5 Class AMQPConnectionException	11
2.5.1 Methods	11
2.5.2 Properties	11
2.6 Class AMQPChannelException	12
2.6.1 Methods	12
2.6.2 Properties	12
3 Module amqplib.client_0_8.abstract_channel	14
3.1 Class AbstractChannel	14
3.1.1 Methods	14
3.1.2 Properties	15
4 Module amqplib.client_0_8.basic_message	16
4.1 Class Message	16
4.1.1 Methods	18
4.1.2 Properties	19
4.1.3 Class Variables	19
5 Module amqplib.client_0_8.channel	20
5.1 Class Channel	20

5.1.1	Methods	20
5.1.2	Properties	38
6	Module amqplib.client_0.8.connection	39
6.1	Class Connection	39
6.1.1	Methods	40
6.1.2	Properties	42
7	Module amqplib.client_0.8.exceptions	43
7.1	Class AMQPException	43
7.1.1	Methods	43
7.1.2	Properties	43
7.2	Class AMQPConnectionException	44
7.2.1	Methods	44
7.2.2	Properties	44
7.3	Class AMQPChannelException	45
7.3.1	Methods	45
7.3.2	Properties	45
8	Module amqplib.client_0.8.method_framing	46
8.1	Class MethodReader	46
8.1.1	Methods	46
8.1.2	Properties	46
9	Module amqplib.client_0.8.serialization	47
9.1	Variables	47
9.2	Class AMQPReader	47
9.2.1	Methods	47
9.2.2	Properties	48
9.3	Class AMQPWriter	49
9.3.1	Methods	49
9.3.2	Properties	50
9.4	Class GenericContent	50
9.4.1	Methods	51
9.4.2	Properties	51
9.4.3	Class Variables	51
10	Module amqplib.client_0.8.transport	52
10.1	Functions	52
10.2	Variables	52
10.3	Class SSLTransport	52
10.3.1	Methods	52
10.3.2	Properties	53
10.4	Class TCPTransport	53
10.4.1	Methods	53
10.4.2	Properties	53
Index		54

1 Package amqplib

1.1 Modules

- **client_0_8**: AMQP Client implementing the 0-8 spec.
(Section 2, p. 4)
 - **abstract_channel**: Code common to Connection and Channel objects.
(Section 3, p. 14)
 - **basic_message**: Messages for AMQP
(Section 4, p. 16)
 - **channel**: AMQP 0-8 Channels
(Section 5, p. 20)
 - **connection**: AMQP 0-8 Connections
(Section 6, p. 39)
 - **exceptions**: Exceptions used by amqplib.client_0_8
(Section 7, p. 43)
 - **method_framing**: Convert between frames and higher-level AMQP methods
(Section 8, p. 46)
 - **serialization**: Convert between bytestreams and higher-level AMQP types.
(Section 9, p. 47)
 - **transport**: Read/Write AMQP frames over network transports.
(Section 10, p. 52)

1.2 Variables

Name	Description
__package__	Value: None

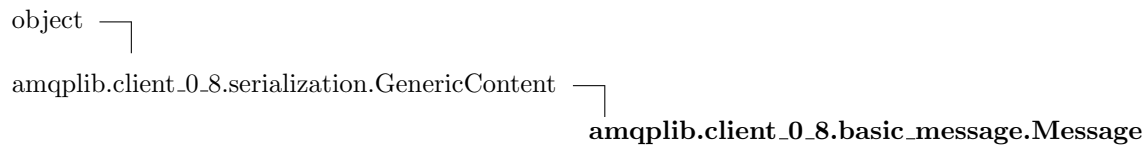
2 Package `amqplib.client_0_8`

AMQP Client implementing the 0-8 spec.

2.1 Modules

- **abstract_channel**: Code common to Connection and Channel objects.
(Section 3, p. 14)
- **basic_message**: Messages for AMQP
(Section 4, p. 16)
- **channel**: AMQP 0-8 Channels
(Section 5, p. 20)
- **connection**: AMQP 0-8 Connections
(Section 6, p. 39)
- **exceptions**: Exceptions used by `amqplib.client_0_8`
(Section 7, p. 43)
- **method_framing**: Convert between frames and higher-level AMQP methods
(Section 8, p. 46)
- **serialization**: Convert between bytestreams and higher-level AMQP types.
(Section 9, p. 47)
- **transport**: Read/Write AMQP frames over network transports.
(Section 10, p. 52)

2.2 Class Message



A Message for use with the `Channel.basic_*` methods.

2.2.1 Methods

```
__init__(self, body='', children=None, **properties)
```

Expected arg types

```
body: string
children: (not supported)
```

Keyword properties may include:

```
content_type: shortstr
    MIME content type

content_encoding: shortstr
    MIME content encoding

application_headers: table
    Message header field table, a dict with string keys,
    and string | int | Decimal | datetime | dict values.

delivery_mode: octet
    Non-persistent (1) or persistent (2)

priority: octet
    The message priority, 0 to 9

correlation_id: shortstr
    The application correlation identifier

reply_to: shortstr
    The destination to reply to

expiration: shortstr
    Message expiration specification

message_id: shortstr
    The application message identifier

timestamp: datetime.datetime
    The message timestamp

type: shortstr
    The message type name

user_id: shortstr
    The creating user id

app_id: shortstr
    The creating application id

cluster_id: shortstr
    Intra-cluster routing identifier
```

Unicode bodies are encoded according to the 'content.encoding' argument. If that's None, it's set to 'UTF-8' automatically.

```
--eq--(self, other)
```

Check if the properties and bodies of this Message and another Message are the same.

Received messages may contain a 'delivery_info' attribute, which isn't compared.

Overrides: amqplib.client_0_8.serialization.GenericContent...eq--

Inherited from amqplib.client_0_8.serialization.GenericContent(Section 9.4)

```
--getattr--(), --ne--()
```

Inherited from object

```
--delattr--(), --format--(), --getattribute--(), --hash--(), --new--(), --reduce--(), --reduce_ex--(),
--repr--(), --setattr--(), --sizeof--(), --str--(), --subclasshook--()
```

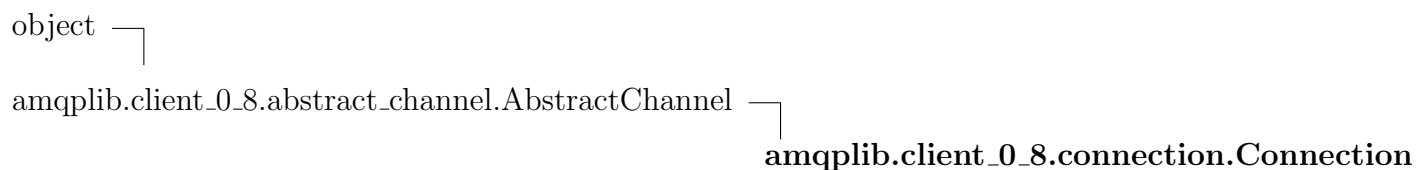
2.2.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>--class--</code>	

2.2.3 Class Variables

Name	Description
PROPERTIES	Value: [('content_type', 'shortstr'), ('content_encoding', 'shor...)

2.3 Class Connection



The connection class provides methods for a client to establish a network connection to a server, and for both peers to operate the connection thereafter.

GRAMMAR:

```

connection      = open-connection *use-connection close-connection
open-connection = C:protocol-header
  
```

```

S:START C:START-OK
*challenge
S:TUNE C:TUNE-OK
C:OPEN S:OPEN-OK | S:REDIRECT
challenge = S:SECURE C:SECURE-OK
use-connection = *channel
close-connection = C:CLOSE S:CLOSE-OK
/ S:CLOSE C:CLOSE-OK

```

2.3.1 Methods

```

__init__(self, host='localhost', userid='guest', password='guest',
login_method='AMQPLAIN', login_response=None, virtual_host='/',
locale='en_US', client_properties=None, ssl=False, insist=False,
connect_timeout=None, **kwargs)

```

Create a connection to the specified host, which should be a 'host[:port]', such as 'localhost', or '1.2.3.4:5672' (defaults to 'localhost', if a port is not specified then 5672 is used)

If login_response is not specified, one is built up for you from userid and password if they are present.

Overrides: object.__init__

```

channel(self, channel_id=None)

```

Fetch a Channel object identified by the numeric channel_id, or create that object if it doesn't already exist.

```
close(self, reply_code=0, reply_text='', method_sig=(0, 0))
```

request a connection close

This method indicates that the sender wants to close the connection. This may be due to internal conditions (e.g. a forced shut-down) or due to an error handling a specific method, i.e. an exception. When a close is due to an exception, the sender provides the class and method id of the method which caused the exception.

RULE:

After sending this method any received method except the Close-OK method MUST be discarded.

RULE:

The peer sending this method MAY use a counter or timeout to detect failure of the other peer to respond correctly with the Close-OK method.

RULE:

When a server receives the Close method from a client it MUST delete all server-side resources associated with the client's context. A client CANNOT reconnect to a context after sending or receiving a Close method.

PARAMETERS:

reply_code: short

The reply code. The AMQ reply codes are defined in AMQ RFC 011.

reply_text: shortstr

The localised reply text. This text can be logged as an aid to resolving issues.

class_id: short

failing method class

When the close is provoked by a method exception, this is the class of the method.

method_id: short

Inherited from amqplib.client_0.8.abstract_channel.AbstractChannel(Section 3.1)

`__enter__()`, `__exit__()`, `wait()`

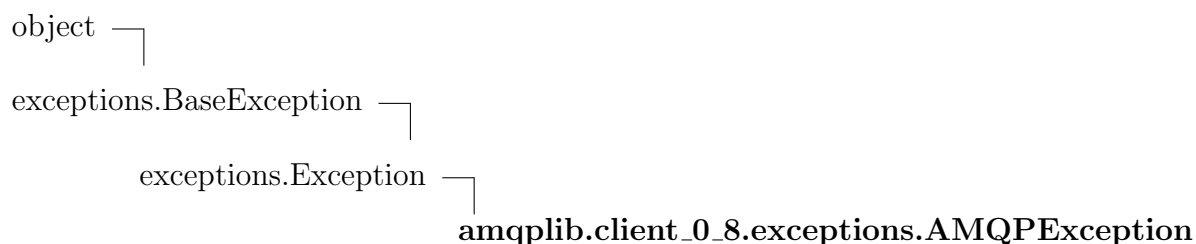
Inherited from object

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`

2.3.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

2.4 Class AMQPException



Known Subclasses: `amqplib.client_0.8.exceptions.AMQPChannelException`, `amqplib.client_0.8.exceptions.AMQPConnectionException`

2.4.1 Methods

<p><code>__init__(self, reply_code, reply_text, method_sig)</code></p> <p><code>x.__init__(...)</code> initializes <code>x</code>; see <code>x.__class__.__doc__</code> for signature</p> <p>Overrides: <code>object.__init__</code> <code>exitit</code>(inherited documentation)</p>

Inherited from exceptions.Exception

`__new__()`

Inherited from exceptions.BaseException

`__delattr__()`, `__getattr__()`, `__getitem__()`, `__getslice__()`, `__reduce__()`, `__repr__()`, `__setattr__()`, `__setstate__()`, `__str__()`, `__unicode__()`

Inherited from object

`__format__()`, `__hash__()`, `__reduce_ex__()`, `__sizeof__()`, `__subclasshook__()`

2.4.2 Properties

Name	Description
<i>Inherited from <code>exceptions.BaseException</code></i>	<code>args</code> , <code>message</code>
<i>Inherited from <code>object</code></i>	<code>__class__</code>

2.5 Class `AMQPConnectionException`



2.5.1 Methods

Inherited from `amqplib.client_0_8.exceptions.AMQPException` (Section 7.1)

`__init__()`

Inherited from `exceptions.Exception`

`__new__()`

Inherited from `exceptions.BaseException`

`__delattr__()`, `__getattr__()`, `__getitem__()`, `__getslice__()`, `__reduce__()`, `__repr__()`,
`__setattr__()`, `__setstate__()`, `__str__()`, `__unicode__()`

Inherited from `object`

`__format__()`, `__hash__()`, `__reduce_ex__()`, `__sizeof__()`, `__subclasshook__()`

2.5.2 Properties

Name	Description
<i>Inherited from exceptions.BaseException</i>	args, message
<i>Inherited from object</i>	<code>__class__</code>

2.6 Class AMQPChannelException



2.6.1 Methods

Inherited from amqplib.client_0.8.exceptions.AMQPException(Section 7.1)

`__init__()`

Inherited from exceptions.Exception

`__new__()`

Inherited from exceptions.BaseException

`__delattr__()`, `__getattr__()`, `__getitem__()`, `__getslice__()`, `__reduce__()`, `__repr__()`,
`__setattr__()`, `__setstate__()`, `__str__()`, `__unicode__()`

Inherited from object

`__format__()`, `__hash__()`, `__reduce_ex__()`, `__sizeof__()`, `__subclasshook__()`

2.6.2 Properties

Name	Description
<i>Inherited from exceptions.BaseException</i>	args, message
<i>Inherited from object</i>	

continued on next page

Name	Description
__class__	

3 Module `amqplib.client_0_8.abstract_channel`

Code common to Connection and Channel objects.

3.1 Class `AbstractChannel`

object 

Known Subclasses: `amqplib.client_0_8.connection.Connection`, `amqplib.client_0_8.channel.Channel`

Superclass for both the Connection, which is treated as channel 0, and other user-created Channel objects.

The subclasses must have a `_METHOD_MAP` class property, mapping between AMQP method signatures and Python methods.

3.1.1 Methods

<p><code>__init__(self, connection, channel_id)</code> <code>x.__init__(...)</code> initializes x; see <code>x.__class__.__doc__</code> for signature Overrides: <code>object.__init__</code> <code>exitit</code>(inherited documentation)</p>

<p><code>__enter__(self)</code> Support for Python ≥ 2.5 'with' statements.</p>

<p><code>__exit__(self, type, value, traceback)</code> Support for Python ≥ 2.5 'with' statements.</p>
--

<p><code>close(self)</code> Close this Channel or Connection</p>
--

<p><code>wait(self, allowed_methods=None)</code> Wait for a method that matches our <code>allowed_methods</code> parameter (the default value of <code>None</code> means match any method), and dispatch to it.</p>

Inherited from object

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`,

`--repr--()`, `--setattr--()`, `--sizeof--()`, `--str--()`, `--subclasshook--()`

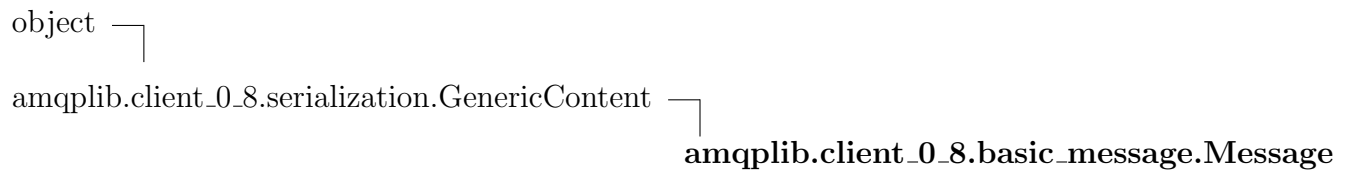
3.1.2 Properties

Name	Description
<i>Inherited from object</i> <code>--class--</code>	

4 Module `amqplib.client_0_8.basic_message`

Messages for AMQP

4.1 Class `Message`



A `Message` for use with the `Channel.basic_*` methods.

4.1.1 Methods

```
__init__(self, body='', children=None, **properties)
```

Expected arg types

```
body: string
children: (not supported)
```

Keyword properties may include:

```
content_type: shortstr
    MIME content type

content_encoding: shortstr
    MIME content encoding

application_headers: table
    Message header field table, a dict with string keys,
    and string | int | Decimal | datetime | dict values.

delivery_mode: octet
    Non-persistent (1) or persistent (2)

priority: octet
    The message priority, 0 to 9

correlation_id: shortstr
    The application correlation identifier

reply_to: shortstr
    The destination to reply to

expiration: shortstr
    Message expiration specification

message_id: shortstr
    The application message identifier

timestamp: datetime.datetime
    The message timestamp

type: shortstr
    The message type name

user_id: shortstr
    The creating user id

app_id: shortstr
```

<code>--eq--(self, other)</code>
Check if the properties and bodies of this Message and another Message are the same.
Received messages may contain a 'delivery_info' attribute, which isn't compared.
Overrides: amqplib.client_0_8.serialization.GenericContent.--eq--

Inherited from amqplib.client_0_8.serialization.GenericContent(Section 9.4)

`--getattr--()`, `--ne--()`

Inherited from object

`--delattr--()`, `--format--()`, `--getattr--()`, `--getattribute--()`, `--hash--()`, `--new--()`, `--reduce--()`, `--reduce_ex--()`, `--repr--()`, `--setattr--()`, `--sizeof--()`, `--str--()`, `--subclasshook--()`

4.1.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>--class--</code>	

4.1.3 Class Variables

Name	Description
PROPERTIES	Value: [('content_type', 'shortstr'), ('content_encoding', 'shor...)

5 Module `amqplib.client_0_8.channel`

AMQP 0-8 Channels

5.1 Class Channel



work with channels

The channel class provides methods for a client to establish a virtual connection - a channel - to a server and for both peers to operate the virtual connection thereafter.

GRAMMAR:

```

channel          = open-channel *use-channel close-channel
open-channel     = C:OPEN S:OPEN-OK
use-channel      = C:FLOW S:FLOW-OK
                  / S:FLOW C:FLOW-OK
                  / S:ALERT
                  / functional-class
close-channel    = C:CLOSE S:CLOSE-OK
                  / S:CLOSE C:CLOSE-OK

```

5.1.1 Methods

```
__init__(self, connection, channel_id=None, auto_decode=True)
```

Create a channel bound to a connection and using the specified numeric `channel_id`, and open on the server.

The `'auto_decode'` parameter (defaults to `True`), indicates whether the library should attempt to decode the body of Messages to a Unicode string if there's a `'content_encoding'` property for the message. If there's no `'content_encoding'` property, or the decode raises an Exception, the plain string is left as the message body.

Overrides: `object.__init__`

```
close(self, reply_code=0, reply_text='', method_sig=(0, 0))
```

request a channel close

This method indicates that the sender wants to close the channel. This may be due to internal conditions (e.g. a forced shut-down) or due to an error handling a specific method, i.e. an exception. When a close is due to an exception, the sender provides the class and method id of the method which caused the exception.

RULE:

After sending this method any received method except Channel.Close-OK MUST be discarded.

RULE:

The peer sending this method MAY use a counter or timeout to detect failure of the other peer to respond correctly with Channel.Close-OK..

PARAMETERS:

reply_code: short

The reply code. The AMQ reply codes are defined in AMQ RFC 011.

reply_text: shortstr

The localised reply text. This text can be logged as an aid to resolving issues.

class_id: short

failing method class

When the close is provoked by a method exception, this is the class of the method.

method_id: short

failing method ID

When the close is provoked by a method exception, this is the ID of the method.

Overrides: amqplib.client_0.8.abstract_channel.AbstractChannel.close

flow(*self*, *active*)

enable/disable flow from peer

This method asks the peer to pause or restart the flow of content data. This is a simple flow-control mechanism that a peer can use to avoid overflowing its queues or otherwise finding itself receiving more messages than it can process. Note that this method is not intended for window control. The peer that receives a request to stop sending content should finish sending the current content, if any, and then wait until it receives a Flow restart method.

RULE:

When a new channel is opened, it is active. Some applications assume that channels are inactive until started. To emulate this behaviour a client MAY open the channel, then pause it.

RULE:

When sending content data in multiple frames, a peer SHOULD monitor the channel for incoming methods and respond to a Channel.Flow as rapidly as possible.

RULE:

A peer MAY use the Channel.Flow method to throttle incoming content data for internal reasons, for example, when exchanging data over a slower connection.

RULE:

The peer that requests a Channel.Flow method MAY disconnect and/or ban a peer that does not respect the request.

PARAMETERS:

active: boolean

start/stop content frames

If True, the peer starts sending content frames. If False, the peer stops sending content frames.

```
access_request(self, realm, exclusive=False, passive=False, active=False,
write=False, read=False)
```

request an access ticket

This method requests an access ticket for an access realm. The server responds by granting the access ticket. If the client does not have access rights to the requested realm this causes a connection exception. Access tickets are a per-channel resource.

RULE:

The realm name MUST start with either "/data" (for application resources) or "/admin" (for server administration resources). If the realm starts with any other path, the server MUST raise a connection exception with reply code 403 (access refused).

RULE:

The server MUST implement the /data realm and MAY implement the /admin realm. The mapping of resources to realms is not defined in the protocol - this is a server-side configuration issue.

PARAMETERS:

realm: shortstr

name of requested realm

RULE:

If the specified realm is not known to the server, the server must raise a channel exception with reply code 402 (invalid path).

exclusive: boolean

request exclusive access

Request exclusive access to the realm. If the server cannot grant this - because there are other active tickets for the realm - it raises a channel exception.

passive: boolean

request passive access

```
exchange_declare(self, exchange, type, passive=False, durable=False,  
auto_delete=True, internal=False, nowait=False, arguments=None,  
ticket=None)
```

declare exchange, create if needed

This method creates an exchange if it does not already exist, and if the exchange exists, verifies that it is of the correct and expected class.

RULE:

The server SHOULD support a minimum of 16 exchanges per virtual host and ideally, impose no limit except as defined by available resources.

PARAMETERS:

`exchange`: shortstr

RULE:

Exchange names starting with "amq." are reserved for predeclared and standardised exchanges. If the client attempts to create an exchange starting with "amq.", the server MUST raise a channel exception with reply code 403 (access refused).

`type`: shortstr

exchange type

Each exchange belongs to one of a set of exchange types implemented by the server. The exchange types define the functionality of the exchange - i.e. how messages are routed through it. It is not valid or meaningful to attempt to change the type of an existing exchange.

RULE:

If the exchange already exists with a different type, the server MUST raise a connection exception with a reply code 507 (not allowed).

RULE:

If the server does not support the requested exchange type it MUST raise a connection exception with a reply code 503 (command invalid).

```
exchange_delete(self, exchange, if_unused=False, nowait=False,  
ticket=None)
```

delete an exchange

This method deletes an exchange. When an exchange is deleted all queue bindings on the exchange are cancelled.

PARAMETERS:

exchange: shortstr

RULE:

The exchange **MUST** exist. Attempting to delete a non-existing exchange causes a channel exception.

if_unused: boolean

delete only if unused

If set, the server will only delete the exchange if it has no queue bindings. If the exchange has queue bindings the server does not delete it but raises a channel exception instead.

RULE:

If set, the server **SHOULD** delete the exchange but only if it has no queue bindings.

RULE:

If set, the server **SHOULD** raise a channel exception if the exchange is in use.

nowait: boolean

do not send a reply method

If set, the server will not respond to the method. The client should not wait for a reply method. If the server could not complete the method it will raise a channel or connection exception.

ticket: short

25

RULE:

The client **MUST** provide a valid access ticket

```
queue_bind(self, queue, exchange, routing_key='', nowait=False,
arguments=None, ticket=None)
```

bind queue to an exchange

This method binds a queue to an exchange. Until a queue is bound it will not receive any messages. In a classic messaging model, store-and-forward queues are bound to a dest exchange and subscription queues are bound to a dest.wild exchange.

RULE:

A server MUST allow ignore duplicate bindings - that is, two or more bind methods for a specific queue, with identical arguments - without treating these as an error.

RULE:

If a bind fails, the server MUST raise a connection exception.

RULE:

The server MUST NOT allow a durable queue to bind to a transient exchange. If the client attempts this the server MUST raise a channel exception.

RULE:

Bindings for durable queues are automatically durable and the server SHOULD restore such bindings after a server restart.

RULE:

If the client attempts to an exchange that was declared as internal, the server MUST raise a connection exception with reply code 530 (not allowed).

RULE:

The server SHOULD support at least 4 bindings per queue, and ideally, impose no limit except as defined by available resources.

26

PARAMETERS:

queue: shortstr

```
queue_declare(self, queue='', passive=False, durable=False,
exclusive=False, auto_delete=True, nowait=False, arguments=None,
ticket=None)
```

declare queue, create if needed

This method creates or checks a queue. When creating a new queue the client can specify various properties that control the durability of the queue and its contents, and the level of sharing for the queue.

RULE:

The server **MUST** create a default binding for a newly-created queue to the default exchange, which is an exchange of type 'direct'.

RULE:

The server **SHOULD** support a minimum of 256 queues per virtual host and ideally, impose no limit except as defined by available resources.

PARAMETERS:

queue: shortstr

RULE:

The queue name **MAY** be empty, in which case the server **MUST** create a new queue with a unique generated name and return this to the client in the Declare-Ok method.

RULE:

Queue names starting with "amq." are reserved for predeclared and standardised server queues. If the queue name starts with "amq." and the passive option is False, the server **MUST** raise a connection exception with reply code 403 (access refused).

passive: boolean

do not create queue

If set, the server will not create the queue. The client can use this to check whether a queue exists without modifying the server state.

```
queue_delete(self, queue='', if_unused=False, if_empty=False,  
nowait=False, ticket=None)
```

delete a queue

This method deletes a queue. When a queue is deleted any pending messages are sent to a dead-letter queue if this is defined in the server configuration, and all consumers on the queue are cancelled.

RULE:

The server SHOULD use a dead-letter queue to hold messages that were pending on a deleted queue, and MAY provide facilities for a system administrator to move these messages back to an active queue.

PARAMETERS:

`queue`: shortstr

Specifies the name of the queue to delete. If the queue name is empty, refers to the current queue for the channel, which is the last declared queue.

RULE:

If the client did not previously declare a queue, and the queue name in this method is empty, the server MUST raise a connection exception with reply code 530 (not allowed).

RULE:

The queue must exist. Attempting to delete a non-existing queue causes a channel exception.

`if_unused`: boolean

delete only if unused

If set, the server will only delete the queue if it has no consumers. If the queue has consumers the server does not delete it but raises a channel exception instead.

RULE:

The server MUST respect the `if-unused` flag when deleting a queue.

`queue_purge(self, queue='', nowait=False, ticket=None)`

purge a queue

This method removes all messages from a queue. It does not cancel consumers. Purged messages are deleted without any formal "undo" mechanism.

RULE:

A call to purge MUST result in an empty queue.

RULE:

On transacted channels the server MUST not purge messages that have already been sent to a client but not yet acknowledged.

RULE:

The server MAY implement a purge queue or log that allows system administrators to recover accidentally-purged messages. The server SHOULD NOT keep purged messages in the same storage spaces as the live messages since the volumes of purged messages may get very large.

PARAMETERS:

queue: shortstr

Specifies the name of the queue to purge. If the queue name is empty, refers to the current queue for the channel, which is the last declared queue.

RULE:

If the client did not previously declare a queue, and the queue name in this method is empty, the server MUST raise a connection exception with reply code 530 (not allowed).

RULE:

The queue must exist. Attempting to purge a non-existing queue causes a channel exception.

nowait: boolean

do not send a reply method

basic_ack(*self*, *delivery_tag*, *multiple*=False)

acknowledge one or more messages

This method acknowledges one or more messages delivered via the Deliver or Get-Ok methods. The client can ask to confirm a single message or a set of messages up to and including a specific message.

PARAMETERS:

delivery_tag: longlong

server-assigned delivery tag

The server-assigned and channel-specific delivery tag

RULE:

The delivery tag is valid only within the channel from which the message was received. I.e. a client MUST NOT receive a message on one channel and then acknowledge it on another.

RULE:

The server MUST NOT use a zero value for delivery tags. Zero is reserved for client use, meaning "all messages so far received".

multiple: boolean

acknowledge multiple messages

If set to True, the delivery tag is treated as "up to and including", so that the client can acknowledge multiple messages with a single method. If set to False, the delivery tag refers to a single message. If the multiple field is True, and the delivery tag is zero, tells the server to acknowledge all outstanding messages.

RULE:

The server MUST validate that a non-zero delivery-tag refers to an delivered message, and raise a channel exception if this is not the case.

basic_cancel(*self*, *consumer_tag*, *nowait*=False)

end a queue consumer

This method cancels a consumer. This does not affect already delivered messages, but it does mean the server will not send any more messages for that consumer. The client may receive an arbitrary number of messages in between sending the cancel method and receiving the cancel-ok reply.

RULE:

If the queue no longer exists when the client sends a cancel command, or the consumer has been cancelled for other reasons, this command has no effect.

PARAMETERS:

consumer_tag: shortstr

consumer tag

Identifier for the consumer, valid within the current connection.

RULE:

The consumer tag is valid only within the channel from which the consumer was created. I.e. a client MUST NOT create a consumer in one channel and then use it in another.

nowait: boolean

do not send a reply method

If set, the server will not respond to the method. The client should not wait for a reply method. If the server could not complete the method it will raise a channel or connection exception.

```
basic_consume(self, queue='', consumer_tag='', no_local=False,  
no_ack=False, exclusive=False, nowait=False, callback=None, ticket=None)
```

start a queue consumer

This method asks the server to start a "consumer", which is a transient request for messages from a specific queue. Consumers last as long as the channel they were created on, or until the client cancels them.

RULE:

The server SHOULD support at least 16 consumers per queue, unless the queue was declared as private, and ideally, impose no limit except as defined by available resources.

PARAMETERS:

queue: shortstr

Specifies the name of the queue to consume from. If the queue name is null, refers to the current queue for the channel, which is the last declared queue.

RULE:

If the client did not previously declare a queue, and the queue name in this method is empty, the server MUST raise a connection exception with reply code 530 (not allowed).

consumer_tag: shortstr

Specifies the identifier for the consumer. The consumer tag is local to a connection, so two clients can use the same consumer tags. If this field is empty the server will generate a unique tag.

RULE:

The tag MUST NOT refer to an existing consumer. If the client attempts to create two consumers with the same non-empty tag the server MUST raise a connection exception with reply code 530 (not allowed).

no_local: boolean

32

do not deliver own messages

```
basic_get(self, queue='', no_ack=False, ticket=None)
```

direct access to a queue

This method provides a direct access to the messages in a queue using a synchronous dialogue that is designed for specific types of application where synchronous functionality is more important than performance.

PARAMETERS:

queue: shortstr

Specifies the name of the queue to consume from. If the queue name is null, refers to the current queue for the channel, which is the last declared queue.

RULE:

If the client did not previously declare a queue, and the queue name in this method is empty, the server **MUST** raise a connection exception with reply code 530 (not allowed).

no_ack: boolean

no acknowledgement needed

If this field is set the server does not expect acknowledgments for messages. That is, when a message is delivered to the client the server automatically and silently acknowledges it on behalf of the client. This functionality increases performance but at the cost of reliability. Messages can get lost if a client dies before it can deliver them to the application.

ticket: short

RULE:

The client **MUST** provide a valid access ticket giving "read" access rights to the realm for the queue.

Non-blocking, returns a message object, or None.

```
basic_publish(self, msg, exchange='', routing_key='', mandatory=False,
immediate=False, ticket=None)
```

publish a message

This method publishes a message to a specific exchange. The message will be routed to queues as defined by the exchange configuration and distributed to any active consumers when the transaction, if any, is committed.

PARAMETERS:

exchange: shortstr

Specifies the name of the exchange to publish to. The exchange name can be empty, meaning the default exchange. If the exchange name is specified, and that exchange does not exist, the server will raise a channel exception.

RULE:

The server **MUST** accept a blank exchange name to mean the default exchange.

RULE:

If the exchange was declared as an internal exchange, the server **MUST** raise a channel exception with a reply code 403 (access refused).

RULE:

The exchange **MAY** refuse basic content in which case it **MUST** raise a channel exception with reply code 540 (not implemented).

routing_key: shortstr

Message routing key

Specifies the routing key for the message. The routing key is used for routing messages depending on the exchange configuration.

mandatory: boolean

34

indicate mandatory routing

This flag tells the server how to react if the message

`basic_qos(self, prefetch_size, prefetch_count, a_global)`

specify quality of service

This method requests a specific quality of service. The QoS can be specified for the current channel or for all channels on the connection. The particular properties and semantics of a qos method always depend on the content class semantics. Though the qos method could in principle apply to both peers, it is currently meaningful only for the server.

PARAMETERS:

`prefetch_size`: long

prefetch window in octets

The client can request that messages be sent in advance so that when the client finishes processing a message, the following message is already held locally, rather than needing to be sent down the channel. Prefetching gives a performance improvement. This field specifies the prefetch window size in octets. The server will send a message in advance if it is equal to or smaller in size than the available prefetch size (and also falls into other prefetch limits). May be set to zero, meaning "no specific limit", although other prefetch limits may still apply. The prefetch-size is ignored if the no-ack option is set.

RULE:

The server **MUST** ignore this setting when the client is not processing any messages - i.e. the prefetch size does not limit the transfer of single messages to a client, only the sending in advance of more messages while the client still has one or more unacknowledged messages.

`prefetch_count`: short

prefetch window in messages

Specifies a prefetch window in terms of whole messages. This field may₃₅ be used in combination with the prefetch-size field; a message will only be sent in advance if both prefetch windows (and those at the channel and connection level) allow it. The prefetch-count is ignored if the no-ack option is set.

basic_recover(*self*, *requeue*=False)

redeliver unacknowledged messages

This method asks the broker to redeliver all unacknowledged messages on a specified channel. Zero or more messages may be redelivered. This method is only allowed on non-transacted channels.

RULE:

The server **MUST** set the redelivered flag on all messages that are resent.

RULE:

The server **MUST** raise a channel exception if this is called on a transacted channel.

PARAMETERS:

requeue: boolean

requeue the message

If this field is False, the message will be redelivered to the original recipient. If this field is True, the server will attempt to requeue the message, potentially then delivering it to an alternative subscriber.

basic_reject(*self*, *delivery_tag*, *requeue*)

reject an incoming message

This method allows a client to reject a message. It can be used to interrupt and cancel large incoming messages, or return untreatable messages to their original queue.

RULE:

The server SHOULD be capable of accepting and process the Reject method while sending message content with a Deliver or Get-Ok method. I.e. the server should read and process incoming methods while sending output frames. To cancel a partially-send content, the server sends a content body frame of size 1 (i.e. with no data except the frame-end octet).

RULE:

The server SHOULD interpret this method as meaning that the client is unable to process the message at this time.

RULE:

A client MUST NOT use this method as a means of selecting messages to process. A rejected message MAY be discarded or dead-lettered, not necessarily passed to another client.

PARAMETERS:

`delivery_tag`: longlong

server-assigned delivery tag

The server-assigned and channel-specific delivery tag

RULE:

The delivery tag is valid only within the channel from which the message was received. I.e. a client MUST NOT receive a message on one channel and then acknowledge it on another.

RULE:

37

The server MUST NOT use a zero value for delivery tags. Zero is reserved for client use, meaning "all messages so far received".

tx_commit (<i>self</i>)

commit the current transaction

This method commits all messages published and acknowledged in the current transaction. A new transaction starts immediately after a commit.
--

tx_rollback (<i>self</i>)

abandon the current transaction

This method abandons all messages published and acknowledged in the current transaction. A new transaction starts immediately after a rollback.

tx_select (<i>self</i>)

select standard transaction mode

This method sets the channel to use standard transactions. The client must use this method at least once on a channel before using the Commit or Rollback methods.
--

Inherited from amqplib.client_0.8.abstract_channel.AbstractChannel(Section 3.1)

`__enter__()`, `__exit__()`, `wait()`

Inherited from object

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`

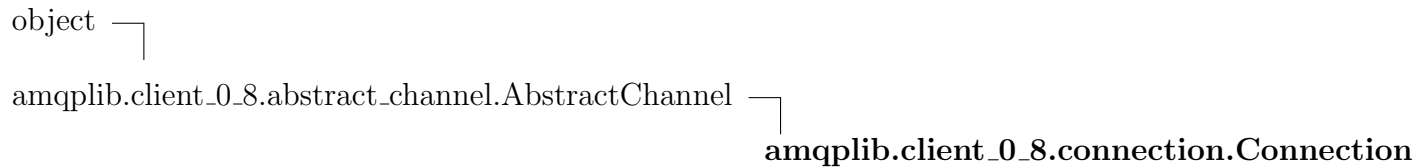
5.1.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

6 Module `amqplib.client_0_8.connection`

AMQP 0-8 Connections

6.1 Class Connection



The connection class provides methods for a client to establish a network connection to a server, and for both peers to operate the connection thereafter.

GRAMMAR:

```

connection          = open-connection *use-connection close-connection
open-connection     = C:protocol-header
                    S:START C:START-OK
                    *challenge
                    S:TUNE C:TUNE-OK
                    C:OPEN S:OPEN-OK | S:REDIRECT
challenge           = S:SECURE C:SECURE-OK
use-connection      = *channel
close-connection    = C:CLOSE S:CLOSE-OK
                    / S:CLOSE C:CLOSE-OK

```

6.1.1 Methods

```
__init__(self, host='localhost', userid='guest', password='guest',  
login_method='AMQPLAIN', login_response=None, virtual_host='/',  
locale='en_US', client_properties=None, ssl=False, insist=False,  
connect_timeout=None, **kwargs)
```

Create a connection to the specified host, which should be a 'host[:port]', such as 'localhost', or '1.2.3.4:5672' (defaults to 'localhost', if a port is not specified then 5672 is used)

If `login_response` is not specified, one is built up for you from `userid` and `password` if they are present.

Overrides: `object.__init__`

```
channel(self, channel_id=None)
```

Fetch a Channel object identified by the numeric `channel_id`, or create that object if it doesn't already exist.

```
close(self, reply_code=0, reply_text='', method_sig=(0, 0))
```

request a connection close

This method indicates that the sender wants to close the connection. This may be due to internal conditions (e.g. a forced shut-down) or due to an error handling a specific method, i.e. an exception. When a close is due to an exception, the sender provides the class and method id of the method which caused the exception.

RULE:

After sending this method any received method except the Close-OK method MUST be discarded.

RULE:

The peer sending this method MAY use a counter or timeout to detect failure of the other peer to respond correctly with the Close-OK method.

RULE:

When a server receives the Close method from a client it MUST delete all server-side resources associated with the client's context. A client CANNOT reconnect to a context after sending or receiving a Close method.

PARAMETERS:

`reply_code`: short

The reply code. The AMQ reply codes are defined in AMQ RFC 011.

`reply_text`: shortstr

The localised reply text. This text can be logged as an aid to resolving issues.

`class_id`: short

failing method class

When the close is provoked by a method exception, this is the class of the method.

`method_id`: short

Inherited from amqplib.client_0_8.abstract_channel.AbstractChannel(Section 3.1)`__enter__()`, `__exit__()`, `wait()`***Inherited from object***`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`,
`__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`**6.1.2 Properties**

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

7.2 Class AMQPConnectionException



7.2.1 Methods

Inherited from amqplib.client_0_8.exceptions.AMQPException(Section 7.1)

`__init__()`

Inherited from exceptions.Exception

`__new__()`

Inherited from exceptions.BaseException

`__delattr__()`, `__getattr__()`, `__getitem__()`, `__getslice__()`, `__reduce__()`, `__repr__()`,
`__setattr__()`, `__setstate__()`, `__str__()`, `__unicode__()`

Inherited from object

`__format__()`, `__hash__()`, `__reduce_ex__()`, `__sizeof__()`, `__subclasshook__()`

7.2.2 Properties

Name	Description
<i>Inherited from exceptions.BaseException</i>	
args, message	
<i>Inherited from object</i>	
__class__	

7.3 Class AMQPChannelException



7.3.1 Methods

Inherited from amqplib.client_0_8.exceptions.AMQPException(Section 7.1)

`--init--()`

Inherited from exceptions.Exception

`--new--()`

Inherited from exceptions.BaseException

`--delattr--()`, `--getattr--()`, `--getitem--()`, `--getslice--()`, `--reduce--()`, `--repr--()`,
`--setattr--()`, `--setstate--()`, `--str--()`, `--unicode--()`

Inherited from object

`--format--()`, `--hash--()`, `--reduce_ex--()`, `--sizeof--()`, `--subclasshook--()`

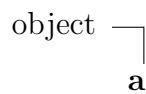
7.3.2 Properties

Name	Description
<i>Inherited from exceptions.BaseException</i>	
args, message	
<i>Inherited from object</i>	
<code>--class--</code>	

8 Module `amqplib.client_0_8.method_framing`

Convert between frames and higher-level AMQP methods

8.1 Class `MethodReader`

object 
`amqplib.client_0_8.method_framing.MethodReader`

Helper class to receive frames from the broker, combine them if necessary with content-headers and content-bodies into complete methods.

Normally a method is represented as a tuple containing (channel, method_sig, args, content).

In the case of a framing error, an `AMQPConnectionException` is placed in the queue.

In the case of unexpected frames, a tuple made up of (channel, `AMQPChannelException`) is placed in the queue.

8.1.1 Methods

<p><code>__init__(self, source)</code> <code>x.__init__(...)</code> initializes x; see <code>x.__class__.__doc__</code> for signature Overrides: <code>object.__init__</code> extit(inherited documentation)</p>

<p><code>read_method(self)</code> <hr/> Read a method from the peer.</p>
--

Inherited from `object`

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`,
`__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`

8.1.2 Properties

Name	Description
<i>Inherited from <code>object</code></i>	
<code>__class__</code>	

9 Module amqplib.client_0_8.serialization

Convert between bytestreams and higher-level AMQP types.

2007-11-05 Barry Pederson <bp@barryp.org>

9.1 Variables

Name	Description
DUMP_CHARS	Value: 'abcdefghijklmnopqrstuvwxzABCDEFGHIJKLMN OPQRSTUVWXYZ0123.
__package__	Value: 'amqplib.client_0_8'

9.2 Class AMQPReader

object └─
amqplib.client_0_8.serialization.AMQPReader

Read higher-level AMQP types from a bytestream.

9.2.1 Methods

__init__ (<i>self</i> , <i>source</i>)
Source should be either a file-like object with a read() method, or a plain (non-unicode) string. Overrides: object.__init__
close (<i>self</i>)
read (<i>self</i> , <i>n</i>)
Read n bytes.
read_bit (<i>self</i>)
Read a single boolean value.
read_octet (<i>self</i>)
Read one byte, return as an integer

read_short (<i>self</i>)
Read an unsigned 16-bit integer

read_long (<i>self</i>)
Read an unsigned 32-bit integer

read_longlong (<i>self</i>)
Read an unsigned 64-bit integer

read_shortstr (<i>self</i>)
Read a utf-8 encoded string that's stored in up to 255 bytes. Return it decoded as a Python unicode object.

read_longstr (<i>self</i>)
Read a string that's up to 2**32 bytes, the encoding isn't specified in the AMQP spec, so just return it as a plain Python string.

read_table (<i>self</i>)
Read an AMQP table, and return as a Python dictionary.

read_timestamp (<i>self</i>)
Read an AMQP timestamp, which is a 64-bit integer representing seconds since the Unix epoch in 1-second resolution. Return as a Python datetime.datetime object, expressed as localtime.

Inherited from object

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`

9.2.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

9.3 Class AMQPWriter

object —
amqplib.client_0_8.serialization.AMQPWriter

Convert higher-level AMQP types to bytestreams.

9.3.1 Methods

__init__(*self*, *dest*=None)

dest may be a file-type object (with a `write()` method). If None then a StringIO is created, and the contents can be accessed with this class's `getvalue()` method.

Overrides: `object.__init__`

close(*self*)

Pass through if possible to any file-like destinations.

flush(*self*)

Pass through if possible to any file-like destinations.

getvalue(*self*)

Get what's been encoded so far if we're working with a StringIO.

write(*self*, *s*)

Write a plain Python string, with no special encoding.

write_bit(*self*, *b*)

Write a boolean value.

write_octet(*self*, *n*)

Write an integer as an unsigned 8-bit value.

write_short(*self*, *n*)

Write an integer as an unsigned 16-bit value.

<code>write_long</code> (<i>self</i> , <i>n</i>)

Write an integer as an unsigned2 32-bit value.
--

<code>write_longlong</code> (<i>self</i> , <i>n</i>)

Write an integer as an unsigned 64-bit value.

<code>write_shortstr</code> (<i>self</i> , <i>s</i>)

Write a string up to 255 bytes long after encoding. If passed a unicode string, encode as UTF-8.
--

<code>write_longstr</code> (<i>self</i> , <i>s</i>)
--

Write a string up to 2**32 bytes long after encoding. If passed a unicode string, encode as UTF-8.
--

<code>write_table</code> (<i>self</i> , <i>d</i>)
--

Write out a Python dictionary made of up string keys, and values that are strings, signed integers, Decimal, datetime.datetime, or sub-dictionaries following the same constraints.

<code>write_timestamp</code> (<i>self</i> , <i>v</i>)
--

Write out a Python datetime.datetime object as a 64-bit integer representing seconds since the Unix epoch.
--

Inherited from object

`--delattr--()`, `--format--()`, `--getattr--()`, `--hash--()`, `--new--()`, `--reduce--()`, `--reduce_ex--()`, `--repr--()`, `--setattr--()`, `--sizeof--()`, `--str--()`, `--subclasshook--()`

9.3.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>--class--</code>	

9.4 Class `GenericContent`

```

object ┌
      │
      └─ amqplib.client_0_8.serialization.GenericContent
  
```

Known Subclasses: `amqplib.client_0_8.basic_message.Message`

Abstract base class for AMQP content. Subclasses should override the `PROPERTIES` attribute.

9.4.1 Methods

<code>__init__(self, **props)</code>

Save the properties appropriate to this AMQP content type in a 'properties' dictionary.

Overrides: <code>object.__init__</code>

<code>__eq__(self, other)</code>

Check if this object has the same properties as another content object.

<code>__getattr__(self, name)</code>

Look for additional properties in the 'properties' dictionary, and if present - the 'delivery_info' dictionary.

<code>__ne__(self, other)</code>

Just return the opposite of <code>__eq__</code>

Inherited from object

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`

9.4.2 Properties

Name	Description
<i>Inherited from object</i>	
<code>__class__</code>	

9.4.3 Class Variables

Name	Description
<code>PROPERTIES</code>	Value: <code>[('dummy', 'shortstr')]</code>

10 Module `amqplib.client_0_8.transport`

Read/Write AMQP frames over network transports.

2009-01-14 Barry Pederson <bp@barryp.org>

10.1 Functions

<code>create_transport</code> (<i>host</i> , <i>connect_timeout</i> , <i>ssl=False</i>)
--

Given a few parameters from the Connection constructor, select and create a subclass of <code>_AbstractTransport</code> .

10.2 Variables

Name	Description
<code>HAVE_PY26_SSL</code>	Value: <code>True</code>
<code>AMQP_PORT</code>	Value: <code>5672</code>
<code>AMQP_PROTOCOL_HEADER</code>	Value: <code>'AMQP\x01\x01\t\x01'</code>
<code>--package--</code>	Value: <code>'amqplib.client_0_8'</code>

10.3 Class `SSLTransport`

object 

`amqplib.client_0_8.transport._AbstractTransport` 

`amqplib.client_0_8.transport.SSLTransport`

Transport that works over SSL

10.3.1 Methods

Inherited from `amqplib.client_0_8.transport._AbstractTransport`

`--del--()`, `--init--()`, `close()`, `read_frame()`, `write_frame()`

Inherited from `object`

`--delattr--()`, `--format--()`, `--getattr--()`, `--hash--()`, `--new--()`, `--reduce--()`, `--reduce_ex--()`, `--repr--()`, `--setattr--()`, `--sizeof--()`, `--str--()`, `--subclasshook--()`

10.3.2 Properties

Name	Description
<i>Inherited from object</i>	
__class__	

10.4 Class TCPTransport

object └

amqplib.client_0_8.transport._AbstractTransport └

amqplib.client_0_8.transport.TCPTransport

Transport that deals directly with TCP socket.

10.4.1 Methods

Inherited from amqplib.client_0_8.transport._AbstractTransport

`__del__()`, `__init__()`, `close()`, `read_frame()`, `write_frame()`

Inherited from object

`__delattr__()`, `__format__()`, `__getattr__()`, `__hash__()`, `__new__()`, `__reduce__()`, `__reduce_ex__()`, `__repr__()`, `__setattr__()`, `__sizeof__()`, `__str__()`, `__subclasshook__()`

10.4.2 Properties

Name	Description
<i>Inherited from object</i>	
__class__	

Index

- amqplib (*package*), 2
 - amqplib.client_0_8 (*package*), 3–12
 - amqplib.client_0_8.abstract_channel (*module*), 13–14
 - amqplib.client_0_8.basic_message (*module*), 15–18
 - amqplib.client_0_8.channel (*module*), 19–37
 - amqplib.client_0_8.connection (*module*), 38–41
 - amqplib.client_0_8.exceptions (*module*), 42–44
 - amqplib.client_0_8.method_framing (*module*), 45
 - amqplib.client_0_8.serialization (*module*), 46–50
 - amqplib.client_0_8.transport (*module*), 51–52