pamqp Documentation

Release 1.6.1

Gavin M. Roy

June 25, 2015

Contents

pamqp is a low level AMQP 0-9-1 frame encoding and decoding library for Python 2 & 3 released under the BSD license.

pamqp is not a end-user client library for talking to RabbitMQ but rather is used by client libraries for marshaling and unmarshaling AMQP frames. All methods should have test coverage and pass in Python 2.6, 2.7, 3.3, and 3.4.

AMQP class/method command class mappings can be found in the *pamqp.specification* module while actual frame encoding and encoding should be run through the *pamqp.frame* module.

CHAPTER 1

Issues

Please report any issues to the Github repo at https://github.com/gmr/pamqp/issues

CHAPTER 2

Source

pamqp source is available on Github at https://github.com/gmr/pamqp

CHAPTER 3

Installation

pamqp is available from the Python Package Index but should generally be installed as a dependency from a client library.

CHAPTER 4

pamqp module documentation

4.1 pamqp.body

The pamqp.body module contains the Body class which is used when unmarshaling body frames. When dealing with content frames, the message body will be returned from the library as an instance of the body class.

class pamqp.body.**ContentBody**(*value=None*)

ContentBody carries the value for an AMQP message body frame

marshal()

Return the marshaled content body. This method is here for API compatibility, there is no special marhsaling for the payload in a content frame.

Return type str|unicode|bytes

unmarshal(*data*)

Apply the data to the object. This method is here for API compatibility, there is no special unmarhsaling for the payload in a content frame.

Return type str|unicode|bytes

4.2 pamqp.decode

AMQP Data Decoder

Functions for decoding data of various types including field tables and arrays

pamqp.decode.**bit**(*value*, *position*)

Decode a bit value

Parameters **value** (*bytes*) – Value to decode

Return tuple bytes used, bool value

Raises ValueError

pamqp.decode.**boolean**(*value*)

Decode a boolean value

Parameters **value** (*bytes*) – Value to decode

Return tuple bytes used, bool

Raises ValueError

pamqp.decode.**by_type**(*value*, *data_type*, *offset=0*)
Decodes values using the specified type

Parameters

- **value** (*bytes*) – Value to decode
- **data_type** (*str*) – type of data to decode

Return tuple bytes consumed, mixed based on field type

pamqp.decode.**byte_array**(*value*)
Decode a byte_array value

Parameters **value** (*bytes*) – Value to decode

Return tuple bytes used, bool

Raises ValueError

pamqp.decode.**decimal**(*value*)
Decode a decimal value

Parameters **value** (*bytes*) – Value to decode

Return tuple bytes used, decimal.Decimal value

Raises ValueError

pamqp.decode.**double**(*value*)
Decode a double value

Parameters **value** (*bytes*) – Value to decode

Return tuple bytes used, float

Raises ValueError

pamqp.decode.**field_array**(*value*)
Decode a field array value

Parameters **value** (*bytes*) – Value to decode

Return tuple bytes used, list

Raises ValueError

pamqp.decode.**field_table**(*value*)
Decode a field array value

Parameters **value** (*bytes*) – Value to decode

Return tuple bytes used, dict

Raises ValueError

pamqp.decode.**floating_point**(*value*)
Decode a floating point value

Parameters **value** (*bytes*) – Value to decode

Return tuple bytes used, float

Raises ValueError

pamqp.decode.**long_int**(*value*)
Decode a long integer value

Parameters **value** (*bytes*) – Value to decode

Return tuple bytes used, int

Raises ValueError

pamqp.decode.**long_long_int**(*value*)
Decode a long-long integer value

Parameters **value** (*bytes*) – Value to decode

Return tuple bytes used, int

Raises ValueError

pamqp.decode.**long_str**(*value*)
Decode a string value

Parameters **value** (*bytes*) – Value to decode

Return tuple bytes used, bytes|str

Raises ValueError

pamqp.decode.**octet**(*value*)
Decode an octet value

Parameters **value** (*bytes*) – Value to decode

Return tuple bytes used, int

Raises ValueError

pamqp.decode.**short_int**(*value*)
Decode a short integer value

Parameters **value** (*bytes*) – Value to decode

Return tuple bytes used, int

Raises ValueError

pamqp.decode.**short_short_int**(*value*)
Decode a short, short integer value

Parameters **value** (*bytes*) – Value to decode

Return tuple bytes used, int

Raises ValueError

pamqp.decode.**short_str**(*value*)
Decode a string value

Parameters **value** (*bytes*) – Value to decode

Return tuple bytes used, unicode|str

Raises ValueError

pamqp.decode.**timestamp**(*value*)
Decode a timestamp value

Parameters **value** (*bytes*) – Value to decode

Return tuple bytes used, struct_time

Raises ValueError

4.3 pamqp.encode

AMQP Data Encoder

Functions for encoding data of various types including field tables and arrays

pamqp.encode.**bit**(*value*, *byte*, *position*)
Encode a bit value

Parameters

- **value** (*int*) – Value to decode
- **byte** (*int*) – The byte to apply the value to
- **position** (*int*) – The position in the byte to set the bit on

Return type tuple of bytes used and a bool value

pamqp.encode.**boolean**(*value*)
Encode a boolean value.

Parameters **value** (*bool*) – Value to encode

Return type bytes

pamqp.encode.**by_type**(*value*, *data_type*)
Takes a value of any type and tries to encode it with the specified encoder.

Parameters

- **value** (*any*) – Value to encode
- **data_type** (*str*) – type of data to encode

Return type bytes

Raises TypeError

pamqp.encode.**byte_array**(*value*)
Encode a bytearray

Parameters **value** (*bytearray*) – Value to encode

Return type bytes

pamqp.encode.**decimal**(*value*)
Encode a decimal.Decimal value.

Parameters **value** (*decimal.Decimal*) – Value to encode

Return type bytes

pamqp.encode.**double**(*value*)
Encode a floating point value as a double

Parameters **value** (*float*) – Value to encode

Return type str

pamqp.encode.**encode_table_value**(*value*)
Takes a value of any type and tries to encode it with the proper encoder

Parameters **value** (*any*) – Value to encode

Return type bytes

Raises TypeError

pamqp.encode.**field_array**(*value*)
Encode a field array from a dictionary.

Parameters **value** (*list*) – Value to encode

Return type bytes

Raises TypeError

pamqp.encode.**field_table**(*value*)
Encode a field table from a dictionary.

Parameters **or None value** (*dict*) – Value to encode

Return type bytes

Raises TypeError

pamqp.encode.**floating_point**(*value*)
Encode a floating point value.

Parameters **value** (*float*) – Value to encode

Return type bytes

pamqp.encode.**long_int**(*value*)
Encode a long integer.

Parameters **value** (*int*) – Value to encode

Return type bytes

pamqp.encode.**long_long_int**(*value*)
Encode a long-long int.

Parameters **or int value** (*long*) – Value to encode

Return type bytes

pamqp.encode.**long_string**(*value*)
Encode a string.

Parameters **value** (*bytes*) – Value to encode

Return type bytes

Raises TypeError

pamqp.encode.**octet**(*value*)
Encode an octet value.

Parameters **value** – Value to encode

Return type bytes

Raises TypeError

pamqp.encode.**short_int**(*value*)
Encode a short integer.

Parameters **value** (*int*) – Value to encode

Return type bytes

Raises TypeError

pamqp.encode.**short_string**(*value*)
Encode a string.

Parameters **value** (*str*) – Value to encode

Return type bytes

Raises TypeError

pamqp.encode.**table_integer**(*value*)
Determines the best type of numeric type to encode value as, preferring the smallest data size first.

Parameters **value** (*int*) – Value to encode

Return type bytes

Raises TypeError

pamqp.encode.**timestamp**(*value*)
Encode a datetime.datetime object or time.struct_time.

Parameters **or time.struct_time value value** (*datetime.datetime*) – Value to encode

Return type bytes

Raises TypeError

4.4 pamqp.frame

Manage the marshaling and unmarshaling of AMQP frames

unmarshal will turn a raw AMQP byte stream into the appropriate AMQP objects from the specification file.

marshal will take an object created from the specification file and turn it into a raw byte stream.

pamqp.frame.**marshal**(*frame_value*, *channel_id*)
Marshal a frame to be sent over the wire.

Parameters

- **frame_value** (*pamqp.specification.Frame or pamqp.heartbeat.Heartbeat*) – The frame object to marshal
- **channel_id** (*int*) – The channel number to send the frame on

Return type str

Raises ValueError

pamqp.frame.**unmarshal**(*data_in*)
Takes in binary data and maps builds the appropriate frame type, returning a frame object.

Parameters **data_in** (*bytes*) – Raw byte stream data

Return type tuple of bytes consumed, channel, and a frame object

Raises specification.FrameError

4.5 pamqp.header

AMQP Header Class Definitions

For encoding AMQP Header frames into binary AMQP stream data and decoding AMQP binary data into AMQP Header frames.

class pamqp.header.**ContentHeader**(*weight=0*, *body_size=0*, *properties=None*)
Represent a content header frame

A Content Header frame is received after a Basic.Deliver or Basic.GetOk frame and has the data and properties for the Content Body frames that follow.

marshal()
Return the AMQP binary encoded value of the frame

unmarshal(*data*)
Dynamically decode the frame data applying the values to the method object by iterating through the attributes in order and decoding them.

Parameters **data** (*bytes*) – The binary encoded method data

Return type int byte count of data used to unmarshal the frame

Raises ValueError

class pamqp.header.**ProtocolHeader**(*major_version=None*, *minor_version=None*, *revision=None*)
Class that represents the AMQP Protocol Header

marshal()
Return the full AMQP wire protocol frame data representation of the ProtocolHeader frame.

Return type str or bytes

unmarshal(*data*)
Dynamically decode the frame data applying the values to the method object by iterating through the attributes in order and decoding them.

Parameters **data** (*bytes*) – The binary encoded method data

Return type int byte count of data used to unmarshal the frame

Raises ValueError

4.6 pamqp.heartbeat

AMQP Heartbeat Frame, used to create new Heartbeat frames for sending to a peer

class pamqp.heartbeat.**Heartbeat**
Heartbeat frame object mapping class. AMQP Heartbeat frames are mapped on to this class for a common access structure to the attributes/data values.

marshal()
Return the binary frame content

Return type str or bytes

4.7 pamqp.specification

The *pamqp.specification* module is auto-generated, created by the `tools/codegen.py` application. It contains all of the information about the protocol that is required for a client library to communicate with RabbitMQ or another AMQP 0-9-1 broker.

The classes inside *pamqp.specification* allow for the automatic marshaling and unmarshaling of AMQP method frames and Basic.Properties. In addition the command classes contain information that designates if they

are synchronous commands and if so, what the expected responses are. Each commands arguments are detailed in the class and are listed in the attributes property.

pamqp.specification also implements AMQP exceptions as Python exceptions so that client libraries can raise these exceptions as is appropriate without having to implement their own extensions for AMQP protocol related issues.
specification.py

Auto-generated AMQP Support Module

WARNING: DO NOT EDIT. To Generate run tools/codegen.py

exception pamqp.specification.**AMQPAccessRefused**
The client attempted to work with a server entity to which it has no access due to security settings.

exception pamqp.specification.**AMQPChannelError**
The client attempted to work with a channel that had not been correctly opened. This most likely indicates a fault in the client layer.

exception pamqp.specification.**AMQPCommandInvalid**
The client sent an invalid sequence of frames, attempting to perform an operation that was considered invalid by the server. This usually implies a programming error in the client.

exception pamqp.specification.**AMQPConnectionForced**
An operator intervened to close the connection for some reason. The client may retry at some later date.

exception pamqp.specification.**AMQPContentTooLarge**
The client attempted to transfer content larger than the server could accept at the present time. The client may retry at a later time.

exception pamqp.specification.**AMQPFrameError**
The sender sent a malformed frame that the recipient could not decode. This strongly implies a programming error in the sending peer.

exception pamqp.specification.**AMQPInternalError**
The server could not complete the method because of an internal error. The server may require intervention by an operator in order to resume normal operations.

exception pamqp.specification.**AMQPInvalidPath**
The client tried to work with an unknown virtual host.

exception pamqp.specification.**AMQPNoConsumers**
When the exchange cannot deliver to a consumer when the immediate flag is set. As a result of pending data on the queue or the absence of any consumers of the queue.

exception pamqp.specification.**AMQPNoRoute**
Undocumented AMQP Soft Error

exception pamqp.specification.**AMQPNotAllowed**
The client tried to work with some entity in a manner that is prohibited by the server, due to security settings or by some other criteria.

exception pamqp.specification.**AMQPNotFound**
The client attempted to work with a server entity that does not exist.

exception pamqp.specification.**AMQPNotImplemented**
The client tried to use functionality that is not implemented in the server.

exception pamqp.specification.**AMQPPreconditionFailed**
The client requested a method that was not allowed because some precondition failed.

exception pamqp.specification.**AMQPResourceError**
The server could not complete the method because it lacked sufficient resources. This may be due to the client creating too many of some type of entity.

exception pamqp.specification.**AMQPResourceLocked**
The client attempted to work with a server entity to which it has no access because another client is working with it.

exception pamqp.specification.**AMQPSyntaxError**
The sender sent a frame that contained illegal values for one or more fields. This strongly implies a programming error in the sending peer.

exception pamqp.specification.**AMQPUnexpectedFrame**
The peer sent a frame that was not expected, usually in the context of a content header and body. This strongly indicates a fault in the peer's content processing.

class pamqp.specification.**Basic**
Work with basic content

The Basic class provides methods that support an industry-standard messaging model.

class **Ack**(*delivery_tag=0*, *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.

class Basic.**Cancel**(*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.

class Basic.**CancelOk**(*consumer_tag=''*)
Confirm a cancelled consumer

This method confirms that the cancellation was completed.

class Basic.**Consume**(*ticket=0*, *queue=''*, *consumer_tag=''*, *no_local=False*, *no_ack=False*, *exclusive=False*, *nowait=False*, *arguments=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 declared on, or until the client cancels them.

class Basic.**ConsumeOk**(*consumer_tag=''*)
Confirm a new consumer

The server provides the client with a consumer tag, which is used by the client for methods called on the consumer at a later stage.

class Basic.**Deliver**(*consumer_tag=''*, *delivery_tag=None*, *redelivered=False*, *exchange=''*, *routing_key=''*)
Notify the client of a consumer message

This method delivers a message to the client, via a consumer. In the asynchronous message delivery model, the client starts a consumer using the Consume method, then the server responds with Deliver methods as and when messages arrive for that consumer.

class Basic.**Get**(*ticket=0*, *queue=''*, *no_ack=False*)
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.

class Basic.**GetEmpty**(*cluster_id=''*)
Indicate no messages available

This method tells the client that the queue has no messages available for the client.

class Basic.**GetOk**(*delivery_tag=None*, *redelivered=False*, *exchange=''*, *routing_key=''*, *message_count=0*)
Provide client with a message

This method delivers a message to the client following a get method. A message delivered by 'get-ok' must be acknowledged unless the no-ack option was set in the get method.

class Basic.**Properties**(*content_type=''*, *content_encoding=''*, *headers=None*, *delivery_mode=None*, *priority=None*, *correlation_id=''*, *reply_to=''*, *expiration=''*, *message_id=''*, *timestamp=None*, *message_type=''*, *user_id=''*, *app_id=''*, *cluster_id=''*)
Content Properties

class Basic.**Publish**(*ticket=0*, *exchange=''*, *routing_key=''*, *mandatory=False*, *immediate=False*)
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.

class Basic.**Qos**(*prefetch_size=0*, *prefetch_count=0*, *global_=False*)
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.

class Basic.**QosOk**
Confirm the requested qos

This method tells the client that the requested QoS levels could be handled by the server. The requested QoS applies to all active consumers until a new QoS is defined.

class Basic.**Recover**(*requeue=False*)
Redeliver unacknowledged messages

This method asks the server to redeliver all unacknowledged messages on a specified channel. Zero or more messages may be redelivered. This method replaces the asynchronous Recover.

class Basic.**RecoverAsync**(*requeue=False*)
Redeliver unacknowledged messages

This method asks the server to redeliver all unacknowledged messages on a specified channel. Zero or more messages may be redelivered. This method is deprecated in favour of the synchronous Recover/RecoverOk.

class Basic.**RecoverOk**
Confirm recovery

This method acknowledges a Basic.Recover method.

class Basic.**Reject**(*delivery_tag=None*, *requeue=True*)
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.

class Basic.**Return**(*reply_code=0*, *reply_text=''*, *exchange=''*, *routing_key=''*)
Return a failed message

This method returns an undeliverable message that was published with the "immediate" flag set, or an unroutable message published with the "mandatory" flag set. The reply code and text provide information about the reason that the message was undeliverable.

class pamqp.specification.**Channel**
Work with channels

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

class **Close**(*reply_code=0*, *reply_text=''*, *class_id=0*, *method_id=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.

class Channel.**CloseOk**
Confirm a channel close

This method confirms a Channel.Close method and tells the recipient that it is safe to release resources for the channel.

class Channel.**Flow**(*active=None*)
Enable/disable flow from peer

This method asks the peer to pause or restart the flow of content data sent by a consumer. 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. It does not affect contents returned by Basic.Get-Ok methods.

class Channel.**FlowOk**(*active=None*)
Confirm a flow method

Confirms to the peer that a flow command was received and processed.

class Channel.**Open**(*out_of_band=''*)
Open a channel for use

This method opens a channel to the server.

class Channel.**OpenOk**(*channel_id=''*)
Signal that the channel is ready

This method signals to the client that the channel is ready for use.

class pamqp.specification.**Connection**
Work with socket connections

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.

class **Blocked**(*reason=''*)
Signal that connection is blocked

This method signals to the client that the connection is blocked by RabbitMQ.

class Connection.**Close**(*reply_code=0*, *reply_text=''*, *class_id=0*, *method_id=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.

class Connection.**CloseOk**
Confirm a connection close

This method confirms a Connection.Close method and tells the recipient that it is safe to release resources for the connection and close the socket.

class Connection.**Open**(*virtual_host='/'*, *capabilities=''*, *insist=False*)
Open connection to virtual host

This method opens a connection to a virtual host, which is a collection of resources, and acts to separate multiple application domains within a server. The server may apply arbitrary limits per virtual host, such as the number of each type of entity that may be used, per connection and/or in total.

class Connection.**OpenOk**(*known_hosts=''*)
Signal that connection is ready

This method signals to the client that the connection is ready for use.

class Connection.**Secure**(*challenge=''*)
Security mechanism challenge

The SASL protocol works by exchanging challenges and responses until both peers have received sufficient information to authenticate each other. This method challenges the client to provide more information.

class Connection.**SecureOk**(*response=''*)
Security mechanism response

This method attempts to authenticate, passing a block of SASL data for the security mechanism at the server side.

class Connection.**Start**(*version_major=0*, *version_minor=9*, *server_properties=None*, *mechanisms='PLAIN'*, *locales='en_US'*)
Start connection negotiation

This method starts the connection negotiation process by telling the client the protocol version that the server proposes, along with a list of security mechanisms which the client can use for authentication.

class Connection.**StartOk**(*client_properties=None*, *mechanism='PLAIN'*, *response=''*, *locale='en_US'*)
Select security mechanism and locale

This method selects a SASL security mechanism.

class Connection.**Tune**(*channel_max=0*, *frame_max=0*, *heartbeat=0*)
Propose connection tuning parameters

This method proposes a set of connection configuration values to the client. The client can accept and/or adjust these.

class Connection.**TuneOk**(*channel_max=0*, *frame_max=0*, *heartbeat=0*)
Negotiate connection tuning parameters

This method sends the client's connection tuning parameters to the server. Certain fields are negotiated, others provide capability information.

class Connection.**Unblocked**
Signal that connection is no longer blocked

This method signals to the client that the connection is no longer blocked by RabbitMQ.

class pamqp.specification.**Exchange**
Work with exchanges

Exchanges match and distribute messages across queues. Exchanges can be configured in the server or declared at runtime.

class **Declare**(*ticket=0*, *exchange=''*, *exchange_type='direct'*, *passive=False*, *durable=False*, *auto_delete=False*, *internal=False*, *nowait=False*, *arguments=None*)
Verify exchange exists, 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.

class Exchange.**DeclareOk**
Confirm exchange declaration

This method confirms a Declare method and confirms the name of the exchange, essential for automatically-named exchanges.

class Exchange.**Delete**(*ticket=0*, *exchange=''*, *if_unused=False*, *nowait=False*)
Delete an exchange

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

class Exchange.**DeleteOk**
Confirm deletion of an exchange

This method confirms the deletion of an exchange.

class pamqp.specification.**Frame**
Base Class for AMQP Methods which specifies the encoding and decoding behavior.

marshal()
Dynamically encode the frame by taking the list of attributes and encode them item by item getting the value form the object attribute and the data type from the class attribute.

Return type str

classmethod **type**(*attr*)
Return the data type for an attribute.

Return type str

unmarshal(*data*)
Dynamically decode the frame data applying the values to the method object by iterating through the attributes in order and decoding them.

Parameters **data** (*str*) – The binary encoded method data

class pamqp.specification.**PropertiesBase**
Provide a base object that marshals and unmarshals the Basic.Properties object values.

classmethod **attributes**()
Return the list of attributes

Return type list

encode_property(*property_name*, *property_value*)
Encode a single property value

Parameters

- **property_name** (*str*) – The property name to encode
- **property_value** (*any*) – The value to encode

marshal()
Take the Basic.Properties data structure and marshal it into the data structure needed for the ContentHeader.

Return type bytes

to_dict()
Return the properties as a dict

Return type dict

classmethod type(*attr*)
Return the data type for an attribute.

Return type str

unmarshal(*flags*, *data*)
Dynamically decode the frame data applying the values to the method object by iterating through the attributes in order and decoding them.

Parameters

- **flags** (*int*) – Flags that indicate if the data has the given property
- **data** (*bytes*) – The binary encoded method data

class pamqp.specification.**Queue**
Work with queues

Queues store and forward messages. Queues can be configured in the server or created at runtime. Queues must be attached to at least one exchange in order to receive messages from publishers.

class Bind(*ticket=0*, *queue=''*, *exchange=''*, *routing_key=''*, *nowait=False*, *arguments=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 direct exchange and subscription queues are bound to a topic exchange.

class Queue.**BindOk**
Confirm bind successful

This method confirms that the bind was successful.

class Queue.**Declare**(*ticket=0*, *queue=''*, *passive=False*, *durable=False*, *exclusive=False*, *auto_delete=False*, *nowait=False*, *arguments=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.

class Queue.**DeclareOk**(*queue=''*, *message_count=0*, *consumer_count=0*)
Confirms a queue definition

This method confirms a Declare method and confirms the name of the queue, essential for automatically-named queues.

class Queue.**Delete**(*ticket=0*, *queue=''*, *if_unused=False*, *if_empty=False*, *nowait=False*)
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.

class Queue.**DeleteOk**(*message_count=0*)
Confirm deletion of a queue

This method confirms the deletion of a queue.

class Queue.**Purge**(*ticket=0*, *queue=''*, *nowait=False*)
Purge a queue

This method removes all messages from a queue which are not awaiting acknowledgment.

class Queue.**PurgeOk**(*message_count=0*)
Confirms a queue purge

This method confirms the purge of a queue.

class Queue.**Unbind**(*ticket=0*, *queue=''*, *exchange=''*, *routing_key=''*, *arguments=None*)
Unbind a queue from an exchange

This method unbinds a queue from an exchange.

class Queue.**UnbindOk**
Confirm unbind successful

This method confirms that the unbind was successful.

class pamqp.specification.**Tx**
Work with transactions

The Tx class allows publish and ack operations to be batched into atomic units of work. The intention is that all publish and ack requests issued within a transaction will complete successfully or none of them will. Servers SHOULD implement atomic transactions at least where all publish or ack requests affect a single queue. Transactions that cover multiple queues may be non-atomic, given that queues can be created and destroyed asynchronously, and such events do not form part of any transaction. Further, the behaviour of transactions with respect to the immediate and mandatory flags on Basic.Publish methods is not defined.

class **Commit**
Commit the current transaction

This method commits all message publications and acknowledgments performed in the current transaction. A new transaction starts immediately after a commit.

class Tx.**CommitOk**
Confirm a successful commit

This method confirms to the client that the commit succeeded. Note that if a commit fails, the server raises a channel exception.

class Tx.**Rollback**
Abandon the current transaction

This method abandons all message publications and acknowledgments performed in the current transaction. A new transaction starts immediately after a rollback. Note that unacked messages will not be automatically redelivered by rollback; if that is required an explicit recover call should be issued.

class Tx.**RollbackOk**
Confirm successful rollback

This method confirms to the client that the rollback succeeded. Note that if an rollback fails, the server raises a channel exception.

class Tx.**Select**
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.

class Tx.**SelectOk**
Confirm transaction mode

This method confirms to the client that the channel was successfully set to use standard transactions.

CHAPTER 5

Version History

See `history`

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

p

A

B

C

D

E

F

H

L

M

O

P

Q

S

T

U