

---

**qpid-bow**

***Release 1.0.2***

**Jul 12, 2018**



---

## Contents

---

<b>1</b>	<b>qpidoBow</b>	<b>1</b>
1.1	qpidoBow package . . . . .	1
	<b>Python Module Index</b>	<b>19</b>



## 1.1 qpid\_bow package

### 1.1.1 Subpackages

**qpid\_bow.cli package**

**Submodules**

**qpid\_bow.cli.connection\_kill module**

`qpid_bow.cli.connection_kill.connection_kill(args)`

`qpid_bow.cli.connection_kill.connection_kill_parser(action)`

**qpid\_bow.cli.message\_receive module**

`qpid_bow.cli.message_receive.message_receive(args)`

`qpid_bow.cli.message_receive.message_receive_parser(action)`

**qpid\_bow.cli.message\_send module**

`qpid_bow.cli.message_send.message_send(args)`

`qpid_bow.cli.message_send.message_send_parser(action)`

### qpid\_bow.cli.qpid\_bow module

```
qpid_bow.cli.qpid_bow.create_command(command: str, helpline: str, action: arg-  
                                     parse._SubParsersAction, subparsers: Iter-  
                                     able[Callable])  
  
qpid_bow.cli.qpid_bow.main()
```

### qpid\_bow.cli.queue\_create module

```
qpid_bow.cli.queue_create.queue_create(args)  
qpid_bow.cli.queue_create.queue_create_parser(action)
```

### qpid\_bow.cli.queue\_delete module

```
qpid_bow.cli.queue_delete.queue_delete(args)  
qpid_bow.cli.queue_delete.queue_delete_parser(action)
```

### qpid\_bow.cli.queue\_purge module

```
qpid_bow.cli.queue_purge.queue_purge(args)  
qpid_bow.cli.queue_purge.queue_purge_parser(action)
```

### qpid\_bow.cli.queue\_reroute module

```
qpid_bow.cli.queue_reroute.queue_reroute(args)  
qpid_bow.cli.queue_reroute.queue_reroute_parser(action)
```

### qpid\_bow.cli.queue\_stats module

```
qpid_bow.cli.queue_stats.queue_stats(args)  
qpid_bow.cli.queue_stats.queue_stats_parser(action)
```

### qpid\_bow.cli.route\_config module

```
qpid_bow.cli.route_config.route_config(args)  
qpid_bow.cli.route_config.route_config_parser(action)
```

### qpid\_bow.cli.route\_dump module

```
qpid_bow.cli.route_dump.route_dump(args)  
qpid_bow.cli.route_dump.route_dump_parser(action)
```

## qpid\_bow.cli.session\_outgoing module

`qpid_bow.cli.session_outgoing.session_outgoing(args)`  
`qpid_bow.cli.session_outgoing.session_outgoing_parser(action)`

## Module contents

### qpid\_bow.management package

#### Submodules

### qpid\_bow.management.connection module

AMQP broker connection management.

`qpid_bow.management.connection.get_connection_ids(server_url: Union[str, NoneType] = None) → list`  
 Retrieve connection ids of all established connections to AMQP broker.

**Parameters** `server_url` – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.

**Returns** List of connections

`qpid_bow.management.connection.kill_connection(connection_id: dict, server_url: Union[str, NoneType] = None)`

Kill connection on AMQP broker.

#### Parameters

- `connection_id` – ID of connection.
- `server_url` – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.

### qpid\_bow.management.exchange module

AMQP broker exchange management.

**class** `qpid_bow.management.exchange.ExchangeType`

Bases: `enum.Enum`

Define type of exchange.

`direct = 'direct'`

`fanout = 'fanout'`

`headers = 'headers'`

`topic = 'topic'`

`qpid_bow.management.exchange.create_binding(exchange_name: str, queue_name: str, binding_name=None, headers_match: dict = None, server_url: Union[str, NoneType] = None)`

Create binding between queue and exchange.

#### Parameters

- **exchange\_name** – Name of exchange.
- **queue\_name** – Name of queue.
- **binding\_name** – Name of binding.
- **headers\_match** – Headers key-value pairs that should be presented on message to match the binding. Only for *headers* exchange type.
- **server\_url** – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.

```
qpido_bow.management.exchange.create_exchange(exchange_name: str, exchange_type:
qpido_bow.management.exchange.ExchangeType
= <ExchangeType.direct: 'direct'>,
durable: bool = True, server_url:
Union[str, NoneType] = None)
```

Create an exchange on the broker.

#### Parameters

- **exchange\_name** – Exchange name.
- **exchange\_type** – *direct, topic, fanout, headers*.
- **durable** – Persist the created exchange on broker restarts.
- **server\_url** – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.

```
qpido_bow.management.exchange.delete_binding(exchange_name: str, queue_name: str,
binding_name: str = None, server_url:
Union[str, NoneType] = None)
```

Delete a binding on the broker.

#### Parameters

- **exchange\_name** – Name of exchange.
- **queue\_name** – Name of queue.
- **binding\_name** – Name of binding.
- **server\_url** – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.

```
qpido_bow.management.exchange.delete_exchange(exchange_name: str, server_url:
Union[str, NoneType] = None)
```

Delete an exchange on the broker.

#### Parameters

- **exchange\_name** – Exchange name.
- **server\_url** – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.

```
qpido_bow.management.exchange.get_binding_keys(exchange_name: str, queue_name: str =
None, server_url: Union[str, NoneType]
= None) → Set[Tuple[[str, str], str]]
```

Retrieve all bindings for specified exchange.

#### Parameters

- **exchange\_name** – Name of exchange.
- **queue\_name** – Name of queue.



- **server\_url** – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.

**Returns** Set of binding keys.

`qpid_bow.management.exchange.get_exchange_bindings(server_url: Union[str, NoneType] = None) → dict`

Retrieve all exchanges and bindings associated with these exchanges.

**Parameters** **server\_url** – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.

**Returns** A dict mapping between exchange it's bindings.

**Return type** dict

### Example

```
>>> get_exchange_bindings()
{'org.apache.qpid.broker:exchange': [{'queue_id': 'org.apache.qpid.
↪broker:queue:examples', 'headers_match': {}}]}
```

`qpid_bow.management.exchange.get_headers_binding_name(exchange: str, queue_name: str, headers_match: dict)`

Generate UUID for exchange, queue and binding.

#### Parameters

- **exchange** – Name of exchange.
- **queue\_name** – Name of queue.
- **headers\_match** – Headers key-value pairs that should be presented on message to match the binding. Only for *headers* exchange type.

**Returns** UUID generated based on specified arguments.

## qpid\_bow.management.queue module

AMQP broker queue management.

`qpid_bow.management.queue.create_queue(queue_name: str, durable: bool = True, auto_delete: bool = False, priorities: int = 0, extra_properties: Union[dict, NoneType] = None, server_url: Union[str, NoneType] = None)`

Create a queue on the AMQP broker.

#### Parameters

- **queue\_name** – Name of queue.
- **durable** – Persist the created queue on broker restarts.
- **auto\_delete** – Delete queue after consumer is disconnected from broker.
- **priorities** – The number of priorities to support.
- **extra\_properties** – Additional properties that will be added during queue creation.
- **server\_url** – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.

`qpid_bow.management.queue.delete_queue` (*queue\_name: str, server\_url: Union[str, NoneType] = None*)

Delete a queue on the AMQP broker.

**Parameters**

- **queue\_name** – Name of queue.
- **server\_url** – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.

`qpid_bow.management.queue.purge_queue` (*queue\_name: str, limit: int = 0, message\_filter: Union[typing.Tuple[str, str], NoneType] = None, server\_url: Union[str, NoneType] = None*)

Purge a queue on the AMQP broker.

**Parameters**

- **queue\_name** – Name of queue.
- **limit** – Limit the amount of messages to purge.
- **message\_filter** – Filter based on property=value.
- **server\_url** – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.

`qpid_bow.management.queue.reroute_queue` (*queue\_name: str, exchange\_name: str, limit: int = 0, message\_filter: Union[typing.Tuple[str, str], NoneType] = None, server\_url: Union[str, NoneType] = None*)

Reroute messages from a queue to an exchange.

**Parameters**

- **queue\_name** – Name of queue.
- **exchange\_name** – Name of exchange.
- **limit** – Limit the amount of messages to reroute.
- **message\_filter** – Filter based on property=value.
- **server\_url** – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.

## qpid\_bow.management.session module

Gather session-related data from AMQP broker.

`qpid_bow.management.session.get_outgoing_sessions_by_address` (*server\_url: Union[str, NoneType] = None*)  
 → MutableMapping[str, list]

Retrieve outgoing sessions from AMQP broker.

**Parameters** **server\_url** – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.

**Returns** A dict mapping between address name and list of sessions.

**Return type** defaultdict

## Example

```
>>> get_outgoing_sessions_by_address()
{'8152f68b-c74a-4d22-8630-a89cf194d067_8152f68b-c74a-4d22-8630-a89cf194d067-
↳2d808664-fe81-4da4-8258-288a7ff531ac': [{'session_id': 'org.apache.qpid.
↳broker:session:0x7fb8bc021ab0', 'transfers': ulong(0)}]}
```

`qpid_bow.management.session.get_sessions(server_url: Union[str, NoneType] = None) → dict`

Retrieve sessions from AMQP broker.

**Parameters** `server_url` – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.

**Returns** A dict mapping between session id and it's address.

**Return type** dict

## Example

```
>>> get_sessions()
{'org.apache.qpid.broker:session:0x7fb8bc021ab0': {'address': '10.0.0.2:34814'}}
```

## qpid\_bow.management.statistics module

Gather statistics from AMQP broker.

`qpid_bow.management.statistics.exchange_statistics(server_url: Union[str, NoneType] = None) → dict`

Retrieve total and dropped amount of messages for exchanges from AMQP broker.

**Parameters** `server_url` – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.

**Returns**

A dict mapping between exchange address and dict with exchange name, total messages count and dropped messages count.

**Return type** dict

## Example

```
>>> exchange_statistics()
{'org.apache.qpid.broker:exchange:': {'name': '', 'total': ulong(236), 'dropped':
↳ulong(0)}}
```

`qpid_bow.management.statistics.gather_statistics(server_url: Union[str, NoneType] = None) → dict`

Retrieve statistics about exchanges and queues from AMQP broker.

Statistics data includes exchanges and queues. Exchange information includes exchange name, total and dropped amount of messages. Queue information includes messages count, depth and bindings to exchange.

**Parameters** `server_url` – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.

**Returns** Exchange and queue statistics.

**Return type** dict

### Example

```
>>> gather_statistics()
{'exchanges': {'org.apache.qpid.broker:exchange:': {'dropped': 0, 'name': '
↳', 'total': 251}}, 'queues': {'org.apache.qpid.broker:queue:examples': {
↳ 'bindings': [{'exchange_id': 'org.apache.qpid.broker:exchange:', 'name':
↳ 'default_route', 'total': 96}], 'depth': 12, 'name': 'examples', 'total': 96}}}
```

```
qpido_bow.management.statistics.queue_statistics(queue_name: Union[str, NoneType]
= None, include_autodelete: bool =
False, server_url: Union[str, None-
Type] = None) → dict
```

Retrieve total messages count and depth for all queues from AMQP broker.

#### Parameters

- **queue\_name** – Name of queue.
- **include\_autodelete** – Include autodelete queues to output.
- **server\_url** – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.

**Returns** A dict mapping between queue address and dict with total messages and queue depth.

**Return type** dict

### Example

```
>>> queue_statistics(queue_name='examples')
{'org.apache.qpid.broker:queue:examples': {'name': 'examples', 'total': 96, 'depth
↳ ': 12}}
```

## Module contents

```
qpido_bow.management.create_QMF2_message() → proton._message.Message
```

Factory function to create a base message for a QMF2 RPC call.

**Returns** A base message for a QMF2 RPC call.

**Return type** Message

```
qpido_bow.management.create_QMF2_method_invoke(object_id: dict, method_name: str, ar-
guments: Mapping[str, Any]) → pro-
ton._message.Message
```

Factory function to create a QMF2 object method call.

#### Parameters

- **object\_id** – Qpid internal object ID.
- **method\_name** – Name of the method to call.
- **arguments** – Mapping with key/value pairs of arguments for the method.

**Returns** A message fully setup with a QMF2 method invoke RPC call.

**Return type** Message

`qpid_bow.management.create_QMF2_query(package_name: str, class_name: str) → proton._message.Message`

Factory function to create a QMF2 object query.

**Parameters**

- **package\_name** – Qpid internal package name to query.
- **class\_name** – Qpid internal class name to query.

**Returns** A message fully setup with a QMF2 object query RPC call.

**Return type** Message

`qpid_bow.management.get_broker_id(server_url: Union[str, NoneType] = None) → dict`

Get the full internal broker ID object.

**Parameters** **server\_url** – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.

**Returns** Full internal broker ID object.

**Return type** dict

`qpid_bow.management.get_object(package_name: str, class_name: str, object_name: str, server_url: Union[str, NoneType] = None) → dict`

Find a raw QMF2 object by type and name.

**Parameters**

- **package\_name** – Qpid internal package name to query.
- **class\_name** – Qpid internal class name to query.
- **object\_name** – Name of the Qpid object to find.
- **server\_url** – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.

**Returns** Raw QMF2 object.

**Return type** dict

`qpid_bow.management.handle_QMF2_exception(message: proton._message.Message)`

Deserialises and raises a QMF2 exception from a reply, in case the QMF2 RPC call failed.

**Parameters** **message** – The QMF2-RPC reply message.

## 1.1.2 Submodules

### 1.1.3 qpid\_bow.asyncio module

**class** `qpid_bow.asyncio.AsyncioReactorHandler(loop=None, handler_base=None)`

Bases: object

Qpid Proton Reactor Global Loop Handler for Python asyncio.

This implementation will setup Qpid Proton's Selectables to use asyncio's writable/readable event handling.

Based on Tornado implementation: [https://qpid.apache.org/releases/qpid-proton-0.18.1/proton/python/examples/proton\\_tornado.py.html](https://qpid.apache.org/releases/qpid-proton-0.18.1/proton/python/examples/proton_tornado.py.html)

#### Parameters

- **loop** – An asyncio event loop
- **handler\_base** – An IO Handler

**on\_reactor\_init** (*event*)

**on\_reactor\_quiesced** (*event*)

**on\_selectable\_final** (*event*)

**on\_selectable\_init** (*event*)

**on\_selectable\_updated** (*event*)

**on\_unhandled** (*name, event*)

**class** qpido\_bow.asyncio.**Container** (\**handlers*, \*\**kwargs*)

Bases: `proton.reactor.Container`

Asyncio event loop based Qpid Reactor container.

**Parameters** \***handlers** – One or more connectors

#### Keyword Arguments

- **handler\_base** – An IO Handler.
- **impl** – Reactor implementation, default is `pn_reactor`.

**create\_receiver** (*context*: `Union[proton._endpoints.Connection, proton._endpoints.Session, proton._url.Url, str]`, *source*=None, *target*=None, *name*=None, *dynamic*=False, *handler*=None, *options*: `Union[proton.reactor.LinkOption, typing.List[proton.reactor.LinkOption]]` = None) → `proton._endpoints.Receiver`

Initiate a link to receive messages (subscription).

#### Parameters

- **context** – One of: created session, connection with or without established session, or url to create session.
- **source** – Source address.
- **target** – Target address.
- **name** – Name of the link.
- **dynamic** – Whether a dynamic AMQP queue should be generated.
- **handler** – Custom handler to handle received message.
- **options** – LinkOptions to further control the attachment.

**Returns** A Qpid Receiver link over which messages are received.

**Return type** *Receiver*

**create\_sender** (*context*: `Union[proton._endpoints.Connection, proton._endpoints.Session, proton._url.Url, str]`, *target*=None, *source*=None, *name*=None, *handler*=None, *tags*=None, *options*: `Union[proton.reactor.LinkOption, typing.List[proton.reactor.LinkOption]]` = None) → `proton._endpoints.Sender`

Initiate a link to send messages.

#### Parameters

- **context** – One of: created session, connection with or without established session, or url to create session.

- **target** – Target address.
- **source** – Source address.
- **name** – Name of the link.
- **handler** – Custom handler to handle received message.
- **tags** –
- **options** – LinkOptions to further control the attachment.

**Returns** A Qpid Sender link over which messages are sent.

**Return type** *Sender*

**run()**

Start Reactor container and begin processing.

**touch()**

Instruct the reactor container to do processing.

You might need to call this to startup new sessions. This is already handled for `create_receiver` and `create_sender`.

### 1.1.4 qpid\_bow.config module

Configure qpid-bow.

`qpid_bow.config.configure(new_config: Mapping)`

Updates global config with provided mapping.

**Parameters** `new_config` – Mapping with config data.

`qpid_bow.config.get_urls(urls: Union[str, NoneType] = None) → List[str]`

Retrieves server urls from one of the sources.

The sources priority comes in the following order: passed arguments, global config, `AMQP_SERVERS` environment variable.

**Parameters** `urls` – Comma-separated urls.

**Returns** Returns list of urls to connect to.

**Return type** `List[str]`

### 1.1.5 qpid\_bow.exc module

Exceptions.

**exception** `qpid_bow.exc.MessageCorrupt`

Bases: `Exception`

Corrupt.

**exception** `qpid_bow.exc.ObjectNotFound(class_name, object_name)`

Bases: `Exception`

No object found.

**exception** `qpid_bow.exc.QMF2Exception(exception_message: str, exception_data: dict)`

Bases: `Exception`

Generic QMF2 exception.

#### Parameters

- **exception\_message** – Message to identify the reason of exception.
- **exception\_data** – Additional data with error code and error text.

**static from\_data** (*exception\_data: dict*)

Try to initialise a specific QMF2Exception based on error code.

**Parameters exception\_data** – Additional data with error code and error text.

**exception** qpido\_bow.exc.QMF2Forbidden (*exception\_data*)

Bases: *qpido\_bow.exc.QMF2Exception*

Forbidden QMF2 call.

**Parameters exception\_data** – Additional data with error code and error text.

**error\_code** = 6

**exception** qpido\_bow.exc.QMF2InvalidParameter (*exception\_data*)

Bases: *qpido\_bow.exc.QMF2Exception*

Invalid parameter is specified.

**Parameters exception\_data** – Additional data with error code and error text.

**error\_code** = 4

**exception** qpido\_bow.exc.QMF2NotFound (*exception\_data*)

Bases: *qpido\_bow.exc.QMF2Exception*

QMF2 object is not found.

**Parameters exception\_data** – Additional data with error code and error text.

**error\_code** = 7

**exception** qpido\_bow.exc.QMF2ObjectExists (*exception\_data*)

Bases: *qpido\_bow.exc.QMF2Exception*

QMF2 object already exists.

**Parameters exception\_data** – Additional data with error code and error text.

**error\_code** = 7

**exception** qpido\_bow.exc.RetriableMessage

Bases: Exception

Release message back to the queue.

**exception** qpido\_bow.exc.TimeoutReached

Bases: Exception

Timeout is reached.

**exception** qpido\_bow.exc.UnroutableMessage

Bases: Exception

Origin message has no reply-to address.

## 1.1.6 qpido\_bow.message module

Message utility methods.



`qpid_bow.message.create_message` (*body: Union[str, bytes, dict, list], properties: Union[dict, NoneType] = None, priority: qpid\_bow.Priority = <Priority.normal: 2>*) → `proton._message.Message`

Utility method to create message with common attributes.

#### Parameters

- **body** – Message body.
- **properties** – Message properties.
- **priority** – Message priority.

**Returns** Created message.

**Return type** Message

`qpid_bow.message.create_reply` (*origin\_message: proton.\_message.Message, result\_data: Union[str, bytes, dict, list]*) → `proton._message.Message`

Create reply to origin message with result data.

Reply messages share the same correlation ID, properties and priority with the exception of being marked as reply.

The address is set to the `reply_to` address from the origin message for usage in a addressless Sender.

#### Parameters

- **origin\_message** – Origin message we are replying to.
- **result\_data** – Message body of the reply.

**Returns** Created reply message.

**Return type** Message

`qpid_bow.message.decode_message` (*data: bytes*) → `proton._message.Message`

Utility method to decode message from bytes.

**Parameters** **data** – Raw AMQP data in bytes.

**Returns** Decoded message.

**Return type** Message

## 1.1.7 qpid\_bow.receiver module

Receive messages from AMQP broker.

```
class qpid_bow.receiver.Receiver (callback: Union[typing.Callable[[proton._message.Message], bool], typing.Callable[[proton._message.Message, proton._delivery.Delivery], bool], typing.Callable[[proton._message.Message], typing.Awaitable[bool]], typing.Callable[[proton._message.Message, proton._delivery.Delivery], typing.Awaitable[bool]]], address: Union[str, NoneType] = None, server_url: Union[str, NoneType] = None, limit: Union[int, NoneType] = None, container_class: Type[Any] = <class 'proton.reactor.Container'>, reconnect_strategy: qpid_bow.ReconnectStrategy = <ReconnectStrategy.backoff: <proton.reactor.Backoff object>>))
```

Bases: `qpid_bow.Connector`

Callback based AMQP message receiver.

#### Parameters

- **callback** – Function to call when new message is received.
- **address** – Name of queue or exchange from where to receive the messages.
- **server\_url** – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.
- **limit** – Limit the amount of messages to receive.
- **container\_class** – Qpid Proton reactor container-class to use.
- **reconnect\_strategy** – Strategy to use on connection drop.

**add\_address** (*address: str*)

Start receiving messages from the given additional address.

**Parameters address** – Queue or exchange address to receive from.

**coroutine handle\_async\_message** (*event*)

**handle\_message** (*event*)

**on\_connection\_opened** (*event: proton.\_events.EventBase*)

**on\_message** (*event*)

Called when a message is received. The message itself can be obtained as a property on the event. For the purpose of referring to this message in further actions (e.g. if explicitly accepting it, the `delivery` should be used, also obtainable via a property on the event.

**on\_start** (*event*)

Handle start event.

**Parameters event** – Reactor init event object with container to connect to.

**on\_timer\_task** (*event: proton.\_events.EventBase*)

Handles the event when a timer is finished.

**Parameters event** – Reactor timer task event object.

**receive** (*timeout: Union[datetime.timedelta, NoneType] = None*)

Start receive loop for up to timeout duration or limit messages.

**Parameters timeout** – Timeout duration to wait for message.

**remove\_address** (*address: str*)

Stop receiving messages from the given address.

**Parameters address** – Queue or exchange address to stop receiving from.

**stop** ()

Stop connection to the AMQP server.

### 1.1.8 qpid\_bow.remote\_procedure module

Remote procedure call handling.

```
class qpid_bow.remote_procedure.RemoteProcedure (callback:
    Union[typing.Callable[[proton._message.Message],
        bool],
        typing.Callable[[proton._message.Message,
            proton._delivery.Delivery], bool],
        typing.Callable[[proton._message.Message],
            typing.Awaitable[bool]],
        typing.Callable[[proton._message.Message,
            proton._delivery.Delivery],
                typing.Awaitable[bool]]],
        address:
            str,
        server_url: Union[str, NoneType] = None,
        reconnect_strategy:
            qpid_bow.ReconnectStrategy =
            <ReconnectStrategy.failover:
            <qpid_bow.NonBackoff object>>)
```

Bases: `qpid_bow.receiver.Receiver`

**This class can be used to handle a simple RPC pattern**, sending a call message and waiting for a reply on a temporary queue and response handling through callbacks.

#### Parameters

- **callback** – Function to call when new message is received.
- **address** – Address of queue or exchange to send the messages to.
- **server\_url** – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.
- **reconnect\_strategy** – Strategy to use on connection drop.

**call** (message: `proton._message.Message`, timeout: `Union[datetime.timedelta, NoneType] = None`)

Send RPC message and wait for call reply. :param message: Message to send to RPC-address. :param timeout: Optional maximum timeout to wait for a reply.

**on\_message** (event)

Called when a message is received. The message itself can be obtained as a property on the event. For the purpose of referring to this message in further actions (e.g. if explicitly accepting it, the `delivery` should be used, also obtainable via a property on the event.

**on\_sendable** (event)

Called when the sender link has credit and messages can therefore be transferred.

**on\_start** (event)

Handle start event.

**Parameters event** – Reactor init event object with container to connect to.

**reply\_to**

Reply to address of our temporary queue.

## 1.1.9 qpid\_bow.sender module

Send messages to AMPQ broker.

```
class qpid_bow.sender.Sender (address: Union[str, NoneType] = None, server_url:
                                Union[str, NoneType] = None, reconnect_strategy:
                                qpid_bow.ReconnectStrategy = <ReconnectStrategy.failover:
                                <qpid_bow.NonBackoff object>>)
```

Bases: `qpid_bow.Connector`

Class to send messages in a batch to an AMQP address.

#### Parameters

- **address** – Address of queue or exchange to send the messages to.
- **server\_url** – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.
- **reconnect\_strategy** – Strategy to use on connection drop.

**on\_sendable** (*event*)

Handles sendable event, sends all the messages in the send\_queue.

**on\_start** (*event*)

Handle start event.

**Parameters event** – Reactor init event object with container to connect to.

**queue** (*messages: Iterable[proton.\_message.Message]*)

Enqueue messages that will be send on calling `send`.

**send** ()

Send queued messages.

## 1.1.10 Module contents

Qpid-bow client framework.

```
class qpid_bow.Connector (server_url: Union[str, NoneType] = None, container_class:
                            Type[proton.reactor.Container] = <class 'proton.reactor.Container'>,
                            reconnect_strategy: qpid_bow.ReconnectStrategy = <ReconnectStrat-
                            egy.backoff: <proton.reactor.Backoff object>>)
```

Bases: `proton.handlers.MessagingHandler`

Initiate and keep connection to AMQP message broker.

#### Parameters

- **server\_url** – Comma-separated list of urls to connect to. Multiple can be specified for connection fallback, the first should be the primary server.
- **container\_class** – Qpid Proton reactor container-class to use.
- **reconnect\_strategy** – Strategy to use on connection drop.

**on\_connection\_closed** (*event: proton.\_events.EventBase*)

Handle close connection event.

**Parameters event** – Connection close event.

**on\_connection\_opened** (*event: proton.\_events.EventBase*)

**on\_start** (*event: proton.\_events.EventBase*)

Handle start event.

**Parameters event** – Reactor init event object with container to connect to.

**on\_transport\_error** (*event: proton.\_events.EventBase*)

Called when some error is encountered with the transport over which the AMQP connection is to be established. This includes authentication errors as well as socket errors.

**run** ()

Start this Connector and setup connection to the AMQP server.

**stop** ()

Stop connection to the AMQP server.

**touch** ()

Instruct the reactor container to do processing.

When running with an alternative container, like the AsyncioContainer, you might need to call this to startup new sessions.

**coroutine wait\_closed** ()

**class** qpido\_bow.NonBackoff

Bases: proton.reactor.Backoff

**next** ()

**class** qpido\_bow.Priority

Bases: enum.Enum

Convenience enum for message priorities.

Qpid supports a configurable amount of priorities for a queue, be sure to have at least 5.

When used on a message and enabled on a queue Qpid will re-order which get send out to a receiver first based on the priority.

**high** = 3

**internal\_low** = 0

**low** = 1

**normal** = 2

**realtime** = 4

**class** qpido\_bow.ReconnectStrategy

Bases: enum.Enum

Define possible reconnect strategies.

**backoff** = <proton.reactor.Backoff object>

**disabled** = False

**failover** = <qpido\_bow.NonBackoff object>

**class** qpido\_bow.RunState

Bases: enum.Enum

Indicate current state of Connector.

**connected** = 5

**failed** = 6

**reconnecting** = 4

**started** = 3

**stopped** = 1

```
stopping = 2
```

### q

- [qpido\\_bow](#), 16
- [qpido\\_bow.asyncio](#), 9
- [qpido\\_bow.cli](#), 3
  - [qpido\\_bow.cli.connection\\_kill](#), 1
  - [qpido\\_bow.cli.message\\_receive](#), 1
  - [qpido\\_bow.cli.message\\_send](#), 1
  - [qpido\\_bow.cli.qpido\\_bow](#), 2
  - [qpido\\_bow.cli.queue\\_create](#), 2
  - [qpido\\_bow.cli.queue\\_delete](#), 2
  - [qpido\\_bow.cli.queue\\_purge](#), 2
  - [qpido\\_bow.cli.queue\\_reroute](#), 2
  - [qpido\\_bow.cli.queue\\_stats](#), 2
  - [qpido\\_bow.cli.route\\_config](#), 2
  - [qpido\\_bow.cli.route\\_dump](#), 2
  - [qpido\\_bow.cli.session\\_outgoing](#), 3
- [qpido\\_bow.config](#), 11
- [qpido\\_bow.exc](#), 11
- [qpido\\_bow.management](#), 8
  - [qpido\\_bow.management.connection](#), 3
  - [qpido\\_bow.management.exchange](#), 3
  - [qpido\\_bow.management.queue](#), 5
  - [qpido\\_bow.management.session](#), 6
  - [qpido\\_bow.management.statistics](#), 7
- [qpido\\_bow.message](#), 12
- [qpido\\_bow.receiver](#), 13
- [qpido\\_bow.remote\\_procedure](#), 14
- [qpido\\_bow.sender](#), 15





**A**

add\_address() (qpido\_bow.receiver.Receiver method), 14  
 AsyncioReactorHandler (class in qpido\_bow.asyncio), 9

**B**

backoff (qpido\_bow.ReconnectStrategy attribute), 17

**C**

call() (qpido\_bow.remote\_procedure.RemoteProcedure method), 15  
 configure() (in module qpido\_bow.config), 11  
 connected (qpido\_bow.RunState attribute), 17  
 connection\_kill() (in module qpido\_bow.cli.connection\_kill), 1  
 connection\_kill\_parser() (in module qpido\_bow.cli.connection\_kill), 1  
 Connector (class in qpido\_bow), 16  
 Container (class in qpido\_bow.asyncio), 10  
 create\_binding() (in module qpido\_bow.management.exchange), 3  
 create\_command() (in module qpido\_bow.cli.qpido\_bow), 2  
 create\_exchange() (in module qpido\_bow.management.exchange), 4  
 create\_message() (in module qpido\_bow.message), 12  
 create\_QMF2\_message() (in module qpido\_bow.management), 8  
 create\_QMF2\_method\_invoke() (in module qpido\_bow.management), 8  
 create\_QMF2\_query() (in module qpido\_bow.management), 9  
 create\_queue() (in module qpido\_bow.management.queue), 5  
 create\_receiver() (qpido\_bow.asyncio.Container method), 10  
 create\_reply() (in module qpido\_bow.message), 13  
 create\_sender() (qpido\_bow.asyncio.Container method), 10

**D**

decode\_message() (in module qpido\_bow.message), 13

delete\_binding() (in module qpido\_bow.management.exchange), 4  
 delete\_exchange() (in module qpido\_bow.management.exchange), 4  
 delete\_queue() (in module qpido\_bow.management.queue), 5  
 direct (qpido\_bow.management.exchange.ExchangeType attribute), 3  
 disabled (qpido\_bow.ReconnectStrategy attribute), 17

**E**

error\_code (qpido\_bow.exc.QMF2Forbidden attribute), 12  
 error\_code (qpido\_bow.exc.QMF2InvalidParameter attribute), 12  
 error\_code (qpido\_bow.exc.QMF2NotFound attribute), 12  
 error\_code (qpido\_bow.exc.QMF2ObjectExists attribute), 12  
 exchange\_statistics() (in module qpido\_bow.management.statistics), 7  
 ExchangeType (class in qpido\_bow.management.exchange), 3

**F**

failed (qpido\_bow.RunState attribute), 17  
 failover (qpido\_bow.ReconnectStrategy attribute), 17  
 fanout (qpido\_bow.management.exchange.ExchangeType attribute), 3  
 from\_data() (qpido\_bow.exc.QMF2Exception static method), 12

**G**

gather\_statistics() (in module qpido\_bow.management.statistics), 7  
 get\_binding\_keys() (in module qpido\_bow.management.exchange), 4  
 get\_broker\_id() (in module qpido\_bow.management), 9  
 get\_connection\_ids() (in module qpido\_bow.management.connection), 3  
 get\_exchange\_bindings() (in module qpido\_bow.management.exchange), 5

get\_headers\_binding\_name() (in module qpidd\_bow.management.exchange), 5  
 get\_object() (in module qpidd\_bow.management), 9  
 get\_outgoing\_sessions\_by\_address() (in module qpidd\_bow.management.session), 6  
 get\_sessions() (in module qpidd\_bow.management.session), 7  
 get\_urls() (in module qpidd\_bow.config), 11

## H

handle\_async\_message() (qpidd\_bow.receiver.Receiver method), 14  
 handle\_message() (qpidd\_bow.receiver.Receiver method), 14  
 handle\_QMF2\_exception() (in module qpidd\_bow.management), 9  
 headers (qpidd\_bow.management.exchange.ExchangeType attribute), 3  
 high (qpidd\_bow.Priority attribute), 17

## I

internal\_low (qpidd\_bow.Priority attribute), 17

## K

kill\_connection() (in module qpidd\_bow.management.connection), 3

## L

low (qpidd\_bow.Priority attribute), 17

## M

main() (in module qpidd\_bow.cli.qpidd\_bow), 2  
 message\_receive() (in module qpidd\_bow.cli.message\_receive), 1  
 message\_receive\_parser() (in module qpidd\_bow.cli.message\_receive), 1  
 message\_send() (in module qpidd\_bow.cli.message\_send), 1  
 message\_send\_parser() (in module qpidd\_bow.cli.message\_send), 1  
 MessageCorrupt, 11

## N

next() (qpidd\_bow.NonBackoff method), 17  
 NonBackoff (class in qpidd\_bow), 17  
 normal (qpidd\_bow.Priority attribute), 17

## O

ObjectNotFound, 11  
 on\_connection\_closed() (qpidd\_bow.Connector method), 16  
 on\_connection\_opened() (qpidd\_bow.Connector method), 16

on\_connection\_opened() (qpidd\_bow.receiver.Receiver method), 14  
 on\_message() (qpidd\_bow.receiver.Receiver method), 14  
 on\_message() (qpidd\_bow.remote\_procedure.RemoteProcedure method), 15  
 on\_reactor\_init() (qpidd\_bow.asyncio.AsyncioReactorHandler method), 10  
 on\_reactor\_quiesced() (qpidd\_bow.asyncio.AsyncioReactorHandler method), 10  
 on\_selectable\_final() (qpidd\_bow.asyncio.AsyncioReactorHandler method), 10  
 on\_selectable\_init() (qpidd\_bow.asyncio.AsyncioReactorHandler method), 10  
 on\_selectable\_updated() (qpidd\_bow.asyncio.AsyncioReactorHandler method), 10  
 on\_sendable() (qpidd\_bow.remote\_procedure.RemoteProcedure method), 15  
 on\_sendable() (qpidd\_bow.sender.Sender method), 16  
 on\_start() (qpidd\_bow.Connector method), 16  
 on\_start() (qpidd\_bow.receiver.Receiver method), 14  
 on\_start() (qpidd\_bow.remote\_procedure.RemoteProcedure method), 15  
 on\_start() (qpidd\_bow.sender.Sender method), 16  
 on\_timer\_task() (qpidd\_bow.receiver.Receiver method), 14  
 on\_transport\_error() (qpidd\_bow.Connector method), 16  
 on\_unhandled() (qpidd\_bow.asyncio.AsyncioReactorHandler method), 10

## P

Priority (class in qpidd\_bow), 17  
 purge\_queue() (in module qpidd\_bow.management.queue), 6

## Q

QMF2Exception, 11  
 QMF2Forbidden, 12  
 QMF2InvalidParameter, 12  
 QMF2NotFound, 12  
 QMF2ObjectExists, 12  
 qpidd\_bow (module), 16  
 qpidd\_bow.asyncio (module), 9  
 qpidd\_bow.cli (module), 3  
 qpidd\_bow.cli.connection\_kill (module), 1  
 qpidd\_bow.cli.message\_receive (module), 1  
 qpidd\_bow.cli.message\_send (module), 1  
 qpidd\_bow.cli.qpidd\_bow (module), 2  
 qpidd\_bow.cli.queue\_create (module), 2  
 qpidd\_bow.cli.queue\_delete (module), 2  
 qpidd\_bow.cli.queue\_purge (module), 2  
 qpidd\_bow.cli.queue\_reroute (module), 2  
 qpidd\_bow.cli.queue\_stats (module), 2  
 qpidd\_bow.cli.route\_config (module), 2  
 qpidd\_bow.cli.route\_dump (module), 2  
 qpidd\_bow.cli.session\_outgoing (module), 3

qpido\_bow.config (module), 11  
 qpido\_bow.exc (module), 11  
 qpido\_bow.management (module), 8  
 qpido\_bow.management.connection (module), 3  
 qpido\_bow.management.exchange (module), 3  
 qpido\_bow.management.queue (module), 5  
 qpido\_bow.management.session (module), 6  
 qpido\_bow.management.statistics (module), 7  
 qpido\_bow.message (module), 12  
 qpido\_bow.receiver (module), 13  
 qpido\_bow.remote\_procedure (module), 14  
 qpido\_bow.sender (module), 15  
 queue() (qpido\_bow.sender.Sender method), 16  
 queue\_create() (in module qpido\_bow.cli.queue\_create), 2  
 queue\_create\_parser() (in module qpido\_bow.cli.queue\_create), 2  
 queue\_delete() (in module qpido\_bow.cli.queue\_delete), 2  
 queue\_delete\_parser() (in module qpido\_bow.cli.queue\_delete), 2  
 queue\_purge() (in module qpido\_bow.cli.queue\_purge), 2  
 queue\_purge\_parser() (in module qpido\_bow.cli.queue\_purge), 2  
 queue\_reroute() (in module qpido\_bow.cli.queue\_reroute), 2  
 queue\_reroute\_parser() (in module qpido\_bow.cli.queue\_reroute), 2  
 queue\_statistics() (in module qpido\_bow.management.statistics), 8  
 queue\_stats() (in module qpido\_bow.cli.queue\_stats), 2  
 queue\_stats\_parser() (in module qpido\_bow.cli.queue\_stats), 2

## R

realtime (qpido\_bow.Priority attribute), 17  
 receive() (qpido\_bow.receiver.Receiver method), 14  
 Receiver (class in qpido\_bow.receiver), 13  
 reconnecting (qpido\_bow.RunState attribute), 17  
 ReconnectStrategy (class in qpido\_bow), 17  
 RemoteProcedure (class in qpido\_bow.remote\_procedure), 14  
 remove\_address() (qpido\_bow.receiver.Receiver method), 14  
 reply\_to (qpido\_bow.remote\_procedure.RemoteProcedure attribute), 15  
 reroute\_queue() (in module qpido\_bow.management.queue), 6  
 RetriableMessage, 12  
 route\_config() (in module qpido\_bow.cli.route\_config), 2  
 route\_config\_parser() (in module qpido\_bow.cli.route\_config), 2  
 route\_dump() (in module qpido\_bow.cli.route\_dump), 2  
 route\_dump\_parser() (in module qpido\_bow.cli.route\_dump), 2  
 run() (qpido\_bow.asyncio.Container method), 11

run() (qpido\_bow.Connector method), 17  
 RunState (class in qpido\_bow), 17

## S

send() (qpido\_bow.sender.Sender method), 16  
 Sender (class in qpido\_bow.sender), 15  
 session\_outgoing() (in module qpido\_bow.cli.session\_outgoing), 3  
 session\_outgoing\_parser() (in module qpido\_bow.cli.session\_outgoing), 3  
 started (qpido\_bow.RunState attribute), 17  
 stop() (qpido\_bow.Connector method), 17  
 stop() (qpido\_bow.receiver.Receiver method), 14  
 stopped (qpido\_bow.RunState attribute), 17  
 stopping (qpido\_bow.RunState attribute), 17

## T

TimeoutReached, 12  
 topic (qpido\_bow.management.exchange.ExchangeType attribute), 3  
 touch() (qpido\_bow.asyncio.Container method), 11  
 touch() (qpido\_bow.Connector method), 17

## U

UnroutableMessage, 12

## W

wait\_closed() (qpido\_bow.Connector method), 17