qpid-bow

Release 1.1.0

Oct 31, 2018

Contents

CHAPTER 1

qpid_bow

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: argparse._SubParsersAction, subparsers: Iterable[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: Optional[str] = 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: Optional[str] = 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: Optional[str] = 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.

qpid_bow.management.exchange.**create_exchange**(*exchange_name: str*, *exchange_type: qpid_bow.management.exchange.ExchangeType = <ExchangeType.direct: 'direct'>*, *durable: bool = True*, *server_url: Optional[str] = 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.

qpid_bow.management.exchange.**delete_binding**(*exchange_name: str*, *queue_name: str*, *binding_name: str = None*, *server_url: Optional[str] = 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.

qpid_bow.management.exchange.**delete_exchange**(*exchange_name: str*, *server_url: Optional[str] = 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.

qpid_bow.management.exchange.**get_binding_keys**(*exchange_name: str*, *queue_name: str = None*, *server_url: Optional[str] = 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: Optional[str] = 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: Optional[dict] = None, server_url: Optional[str] = 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: Optional[str] = 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: Optional[Tuple[str, str]] = None*, *server_url: Optional[str] = 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: Optional[Tuple[str, str]] = None*, *server_url: Optional[str] = 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: Optional[str] = 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: Optional[str] = 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: Optional[str] = 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: Optional[str] = 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': ulong(0), 'name': '
↪', 'total': ulong(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}}}
```

qpid_bow.management.statistics.**queue_statistics**(*queue_name: Optional[str] = None*, *include_autodelete: bool = False*, *server_url: Optional[str] = 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

qpid_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

qpid_bow.management.**create_QMF2_method_invoke**(*object_id: dict*, *method_name: str*, *arguments: Mapping[str, Any]*) → proton._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: Optional[str] = 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: Optional[str] = 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

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 qpid_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, 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** – Wether 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, 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**(*argument_urls: Optional[str] = None*) → List[str]
Retrieves server argument_urls from one of the sources.

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

Parameters **argument_urls** – Comma-separated argument_urls.

Returns Returns list of argument_urls to connect to.

Return type List[str]

qpid_bow.config.**process_url**(*url: str*) → str
Processes a URL for usage with Qpid Proton.

- ActiveMQ amqp+ssl scheme is replaced with amqps.
- Adds username and password from config.

Parameters **url** – Input URL.

Returns Processed URL.

Return type 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 qpid_bow.exc.**QMF2Forbidden**(*exception_data*)
Bases: *qpid_bow.exc.QMF2Exception*

Forbidden QMF2 call.

Parameters **exception_data** – Additional data with error code and error text.

error_code = 6

exception qpid_bow.exc.**QMF2InvalidParameter**(*exception_data*)
Bases: *qpid_bow.exc.QMF2Exception*

Invalid parameter is specified.

Parameters **exception_data** – Additional data with error code and error text.

error_code = 4

exception qpid_bow.exc.**QMF2NotFound**(*exception_data*)
Bases: *qpid_bow.exc.QMF2Exception*

QMF2 object is not found.

Parameters **exception_data** – Additional data with error code and error text.

error_code = 7

exception qpid_bow.exc.**QMF2ObjectExists**(*exception_data*)
Bases: *qpid_bow.exc.QMF2Exception*

QMF2 object already exists.

Parameters **exception_data** – Additional data with error code and error text.

error_code = 7

exception qpid_bow.exc.**RetriableMessage**
Bases: Exception

Release message back to the queue.

exception qpid_bow.exc.**TimeoutReached**
Bases: Exception

Timeout is reached.

exception qpid_bow.exc.**UnroutableMessage**
Bases: Exception

Origin message has no reply-to address.

1.1.6 qpid_bow.message module

Message utility methods.

qpid_bow.message.**create_message**(*body: Union[str, bytes, dict, list], properties: Optional[dict] = 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[Callable[proton._message.Message, bool], Callable[[proton._message.Message, proton._delivery.Delivery], bool], Callable[proton._message.Message, Awaitable[bool]], Callable[[proton._message.Message, proton._delivery.Delivery], Awaitable[bool]]], address: Optional[str] = None, server_url: Optional[str] = None, limit: Optional[int] = 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: Optional[datetime.timedelta] = 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[Callable[proton._message.Message, bool], Callable[[proton._message.Message, proton._delivery.Delivery], bool], Callable[proton._message.Message, Awaitable[bool]], Callable[[proton._message.Message, proton._delivery.Delivery], Awaitable[bool]]], address: str, server_url: Optional[str] = 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: Optional[datetime.timedelta] = 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: Optional[str] = None*, *server_url: Optional[str] = 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: Optional[str] = None*, *container_class: Type[proton.reactor.Container] = <class 'proton.reactor.Container'>*, *reconnect_strategy: qpid_bow.ReconnectStrategy = <ReconnectStrategy.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 qpid_bow.**NonBackoff**
Bases: proton.reactor.Backoff

next()

class qpid_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 qpid_bow.**ReconnectStrategy**
Bases: enum.Enum

Define possible reconnect strategies.

backoff = <proton.reactor.Backoff object>

disabled = False

failover = <qpid_bow.NonBackoff object>

class qpid_bow.**RunState**
Bases: enum.Enum

Indicate current state of Connector.

connected = 5

```
    failed = 6
    reconnecting = 4
    started = 3
    stopped = 1
    stopping = 2
```

Python Module Index

q

Index

H

I

K

L

M

N

O

P

Q

R

S

T

U

W