Red Hat Enterprise MRG 3 Messaging Installation and Configuration Guide

Install and Configure the Red Hat Enterprise MRG Messaging Server

Red Hat Customer Content Services

Red Hat Enterprise MRG 3 Messaging Installation and Configuration Guide

Install and Configure the Red Hat Enterprise MRG Messaging Server

Red Hat Customer Content Services

Legal Notice

Abstract

Installation and configuration of the Red Hat Enterprise Messaging Server.

Table of Contents

MRG 3 Overview

1. The Top Six Differences between MRG Messaging 2 and 3

These are the most significant differences between MRG 2 and MRG 3:

1. The broker and the C++ messaging library (**`qpid::messaging`**) now offer amqp1.0 support via the Apache Proton library (note that transactions are not yet available over amqp1.0).
2. Clustering has been replaced with a new High Availability implementation.
3. Queue Threshold alerts are now edge-triggered, rather than level-triggered. This improves alert rate limiting.
4. The flow-to-disk implementation has been changed to *disk-paged queues* to more efficiently use memory.
5. The **`ring-strict`** limit policy has been dropped.
6. The messaging journal has been replaced with a new implementation - the dynamically-expanding *Linear Store*.

See Also:

- Section 1.3, “MRG 2 Features - Where Are They Now?”
- Section 4.9.2, “Disk-paged Queues”
- Section 4.9.5, “Queue Threshold Alerts (Edge-triggered)”

Report a bug

Chapter 1. Quickly Install MRG Messaging

1.1. The Messaging Server

1.1.1. The Messaging Server

MRG Messaging is an Enterprise-grade tested and supported messaging server based on the Apache Qpid project. The MRG Messaging Server uses an enterprise-grade version of the Apache Qpid broker to provide messaging services.

See Also:

- Section 1.2.3, “Messaging Broker Memory Requirements”

Report a bug

1.1.2. Messaging Broker

The *messaging broker* is the server that stores, forwards, and distributes messages. Red Hat Enterprise Messaging uses the Apache Qpid C++ broker.

Report a bug

1.1.3. Install MRG-M 3 Messaging Server on Red Hat Enterprise Linux 6

1. If you are using RHN classic management for your system, subscribe your system to the base channel for Red Hat Enterprise Linux 6.
2. Additionally, subscribe to the available MRG Messaging software channels relevant to your installation and requirements:

   MRG Messaging Software Channels

   Base Channel

   Subscribe to the **`Additional Services Channels for Red Hat Enterprise Linux 6`** / **`MRG Messaging v.3 (for RHEL-6 Server)`** channel to enable full MRG Messaging Platform installations.

   High Availability Channel

   Subscribe to the **`Additional Services Channels for Red Hat Enterprise Linux 6 / RHEL Server High Availability`** channel to enable High Availability installations.
3. Install the MRG Messaging server and client using the following commands:

Note

If only Messaging Client support is required go directly to Step 4.

MRG Messaging Server and Client

Install the "MRG Messaging" group using the following **yum** command (as root):

```
yum groupinstall "MRG Messaging"
```

High Availability Support

If High Availability support is required, install the package using the following yum command (as root):

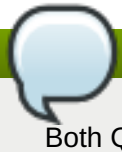
```
yum install qpid-cpp-server-ha
```

4. **Alternative: Install Messaging Client Support Only**

   If only messaging client support is required, install the "Messaging Client Support" group using the following **yum** command (as root):

   ```
   yum groupinstall "Messaging Client Support"
   ```

   You do not need to install this group if you have already installed the "MRG Messaging" group. It is included by default.



Note

Both Qpid JMS AMQP 0.10 and 1.0 clients require Java 1.7 to run. Ensure the Java version installed on your system is 1.7 or higher.

Report a bug

1.1.4. Upgrade a MRG Messaging 2 Server to MRG Messaging 3

Upgrading from MRG Messaging 2 to 3 requires a number of configuration changes in addition to changing RHN channels and installing packages.

1. If you are using RHN classic management for your system, subscribe your system to the base channel for Red Hat Enterprise Linux 6.
2. Remove incompatible components. Run the following command as root:

   ```
   yum erase qpid-cpp-server-cluster sesame cumin cumin-messaging python-wallaby
   ```

3. Unsubscribe the system from the MRG v2 channels.
4. Additionally, subscribe to the available MRG Messaging software channels relevant to your installation and requirements:

   MRG Messaging Software Channels

   Base Channel

   Subscribe to the **Additional Services Channels for Red Hat Enterprise Linux 6** / **MRG Messaging v.3 (for RHEL-6 Server)** channel to enable full MRG Messaging Platform installations.

High Availability Channel

Subscribe to the **Additional Services Channels for Red Hat Enterprise Linux 6 / RHEL Server High Availability** channel to enable High Availability installations.

5. Update the MRG Messaging server and client using the following commands:

Note

If only Messaging Client support is required go directly to Step Six.

MRG Messaging Server and Client

Update the "MRG Messaging" group using the following **yum** command (as root):

```
yum groupinstall "MRG Messaging"
```

High Availability Support

If High Availability support is required, update the package using the following yum command (as root):

```
yum install qpid-cpp-server-ha
```

6. If only messaging client support is required, update the "Messaging Client Support" group using the following **yum** command (as root):

```
yum groupinstall "Messaging Client Support"
```

You do not need to update this group if you have already updated the "MRG Messaging" group. It is included by default.

Note

Both Qpid JMS AMQP 0.10 and 1.0 clients require Java 1.7 to run. Ensure the Java version installed on your system is 1.7 or higher.

See Also:

- Section 1.3, “MRG 2 Features - Where Are They Now?”
- Section 1.4, “Application Migration”

Report a bug

1.1.5. Linearstore Custom Broker EFP Partitions

MRG-M 3.2 introduces an upgraded directory structure for Empty File Pool (EFP) broker partitions, which allows you to specify unique EFP partitions and their sizes.

This feature allows EFPs to be established on different media, and for queues to be able to choose which

partition to use depending on their performance requirements. For example, queues with high throughput and low latency requirements can now be established on more expensive solid state media, while low throughput noncritical queues can be directed to use regular rotating magnetic media.

The new layout allows both the old and new stores to co-exist in mutually exclusive locations in the store directory, which provides the ability to back out of an upgrade if required.

See Also:

- Section 3.3.6, “Persistence Options”
- Section 1.1.6, “Upgrade a MRG Messaging 3.1 Server to MRG Messaging 3.2”

Report a bug

1.1.6. Upgrade a MRG Messaging 3.1 Server to MRG Messaging 3.2

Because of the changes to EFP described in Section 1.1.5, “Linearstore Custom Broker EFP Partitions” some specific requirements exist when upgrading from MRG-M 3.1 to 3.2.

Procedure 1.1. How to Upgrade MRG Messaging 3.1 to 3.2

1. Verify that all required software channels are still correctly subscribed to in Section 1.1.3, “Install MRG-M 3 Messaging Server on Red Hat Enterprise Linux 6”
2. Stop the server by doing one of the following:
    a. Press **Ctrl**+**C** to shutdown the server correctly if started from the command line.
    b. Run **service qpidd stop** to stop the service correctly.
3. Run **sudo yum update qpid-cpp-server-ha** to upgrade to the latest packages.
4. 

    Important

    If you intend to set up custom EFP partitions, complete the steps in Procedure 1.2, “How To Manually Upgrade Linearstore EFP to the New Partitioning Structure” before completing this step.

    Restart the server by running **qpidd** or **service qpidd start** depending on requirements.

If it is not possible to cleanly shut down a MRG-M broker prior to upgrade, the Linearstore EFP files must be manually upgraded to the new structure, and linked correctly.

As part of the Linearstore partition changes, a new directory structure exists.

Directory Changes

qls/dat

This directory is now **qls/dat2**. There is no other change other than the directory name.

qls/tpl

This directory is now **qls/tpl2**.

The journal files previously stored in this directory are now links to journal files. The actual files now reside in **qls/p*NNN*/efp/*[size]*k/in_use** directory in the EFP. This allows the files to be contained within the partition in which the EFP exists.

qls/jrnl

This directory is now **qls/jrnl2**, and contains the *[queue-name]* directories.

The *[queue-name]* directories previously stored in **qls/jrnl** are now links to journal directories. The actual directories now reside in **qls/p*NNN*/efp/*[size]*k/in_use** directory in the EFP. This allows the directories to be contained within the partition in which the EFP exists.

qls/p*NNN*/efp/*[size]*k

Directories of this type now contain an **/in_use** and **/returned** subdirectory, along with the empty files.

p*NNN* relates to the broker partition ID, which is set on the command line using the **--efp-partition** parameter.

***[size]*k** is the size in MiB of the broker partition, which is set on the command line using the **--efp-file-size** parameter.

To ensure data integrity upon live upgrade (where the broker can not be shut down), the new directory structure will not recover a previous store. You must upgrade the store contents manually after taking precautions to back up the store contents.

Note

It is recommended that customers start with a clean store and recreate queues as needed. Only perform an upgrade if:

- You have queues that cannot be recreated.
- There is message data that cannot be expunged before the upgrade.

Example 1.1. Old directory structure

```
qls
    ├── dat (contains Berkeley DB database files)
    ├── p001
    │   └── efp
    │       └── 2028k (contains empty/returned journal files)
    ├── jrnl
    │   ├── queue_1 (contains in-use journal files belonging to queue_1)
    │   ├── queue_2 (contains in-use journal files belonging to queue_2)
    │   ├── queue_3 (contains in-use journal files belonging to queue_3)
    │   ...
    └── tpl (contains in-use journal files belonging to the TPL)
```

Possible variations

- It is possible to use any number of different EFP file sizes, and there may be a number of other directories besides the default of 2048k.
- It is possible to have a number of different partition directories, but in the old Linearstore, these don't perform any useful function other than providing a separate directory for EFP files. These directories must be named **p*NNN***, where *NNN* is a 3-digit number. The partition numbers need not be sequential.

Example 1.2. New directory structure

```
qls
   ├── dat2 (contains Berkeley DB database files)
   ├── p001
   │   └── efp
   │      └── 2028k (contains empty/returned journal files)
   │         ├── in_use (contains in-use journal files)
   │         └── returned (contains files recently returned from being in-
use, but not yet processed before being returned to the 2048k directory)
   │
   ├── jrnl2
   │   ├── queue_1 (contains in-use journal files belonging to queue_1)
   │   ├── queue_2 (contains in-use journal files belonging to queue_2)
   │   ├── queue_3 (contains in-use journal files belonging to queue_3)
   │   ...
   └── tpl2 (contains in-use journal files belonging to the TPL)
```

Note

The database and journal directories are mutually exclusive. It is recommended that the old structure and journals/files be left in place alongside the new structure until the success of the upgrade is confirmed. It also allows that if the upgrade is rolled back to the previous version, the store will continue to operate using the old directory structure.

Procedure 1.2. How To Manually Upgrade Linearstore EFP to the New Partitioning Structure

1. Create new directories **qls/dat2**.

   ```
   # mkdir dat2
   ```

2. Copy the contents of the **Berkeley DB** database from **qls/dat** to the new **qls/dat2** directory.

   ```
   # cp dat/* dat2/
   ```

3. For each EFP directory in **qls/p*NNN*/efp/*[size]*k**, add 2 additional subdirectories;

   a. *in_use*

   ```
   # mkdir p001/efp/2048k/in_use
   ```

   b. *returned*

```
# mkdir p001/efp/2048k/returned
```

By default, there is only one partition; **qls/p001**, and only one EFP size; **2048k**.

4. Create a **jrnl2** directory.

```
# mkdir jrnl2
```

For each directory in the old **jrnl** directory (each of which is named for an existing queue), create an identically named directory in the new **jrnl2** directory.

```
# mkdir jrnl2/[queue-name-1]
# mkdir jrnl2/[queue-name-2]
...
```

You can list the directories present in the **jrnl2** directory with the following command:

```
# dir jrnl
```

5. Each journal file must be first copied to the **in_use** directory of the correct partition directory with correct *efp size* directory. Then a link must be created to this journal file in the new **jrnl2/*[queue-name]*** directory.

Two pieces of information are needed for every journal file:

a. Which partition it originated from.

b. Which size within that partition it is.

The default setting is a single partition number (in directory **qls/p001**), and a single EFP size of **2048k** (which is the approximate size of each journal file). If the old directory structure has only these defaults, then proceed as follows:

a. For each queue in **qls/jrnl**, note the journal files present. Once they are moved, it will be difficult to distinguish which journal files are from which queue as other journal files from other queues will also be present.

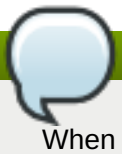
```
# ls -la jrnl/queue-name/*
```

b. Copy all the journal files from the old queue directory into the partition's **2048k in_use** directory.

```
# cp jrnl/queue-name/* p001/efp/2048k/in_use/
```

c. Finally, create a symbolic link to these files in the new queue directory created in step 3 above. This step requires the names of the files copied in step b. above.

```
# ln -s
/abs_path_to/qls/p001/efp/2048k/in_use/journal_1_file_name.jrnl
jrnl2/queue-name/
# ln -s
/abs_path_to/qls/p001/efp/2048k/in_use/journal_2_file_name.jrnl
jrnl2/queue-name/
...
```



Note

When creating a symlink, use an absolute path to the source file.

d. Repeat the previous step for each journal file in queue.

If more than one partition exists, it is important to know which journal files belong to which partition.

You can inspect a hexdump of the file header for each journal file to obtain this information. Note the 2-byte value at offset 26 (0x1a):

```
# hexdump -Cn 4096 path/to/uuid.jrnl
00000000  51 4c 53 66 02 00 00 00  1c 62 0c f1 e2 4c 42 0d
|QLSf.....b...LB.|
00000010  5a 6b 00 00 00 00 00 00  01 00 01 00 00 00 00 00
|Zk..............|
00000020  00 02 00 00 00 00 00 00  00 10 00 00 00 00 00 00
|................|
00000030  34 63 b9 54 00 00 00 00  8e 61 ef 2c 00 00 00 00
|4c.T.....a.,....|
00000040  2f 00 00 00 00 00 00 00  08 00 54 70 6c 53 74 6f
|/.........TplSto|
00000050  72 65 00 00 00 00 00 00  00 00 00 00 00 00 00 00
|re..............|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
|................|
```

In the event that there are several size directories in **p*NNN*/efp/** directory, it is necessary to consider the size of the files being copied in step b. above, and ensure that they are copied to the **in_use** directory of correct efp size.

Example 1.3. More than one size in use in a partition

```
qls
 └── jrnl
      ├── queue-1
      │    └──jrnl1_file.jrnl  (size 2101248)
      └── queue-2
           └──jrnl2_file.jrnl  (size 4198400)
```

Assuming that both these files belong to partition **p*NNN***, then **jrnl1_file.jrnl** will be copied to the new **p*NNN*/efp/2048k/** directory, and **jrnl2_file.jrnl** will be copied to the new pNNN/efp/4096k/ directory.

6. The Transaction Prepared List (TPL) is a special queue which records transaction prepare and commit/abort boundaries for a transaction. In the new store, it is located in a new directory called **tpl2**.

a. Create the **tpl2** directory:

```
# mkdir tpl2
```

b. Repeat the process described in step 4 above, except that the journal files are located in the **tpl** directory, and the symlinks must be created in the new **tpl2** directory:

   a. List current journal files:

   ```
   # ls -la tpl
   ```

   b. Copy journal files to from the **tpl** directory to the correct **p*NNN*/efp/*[size]*k/in_use** alongside other files copied as part of step 4 above.

   ```
   # cp tpl/* p001/efp/2048k/in_use/
   ```

   c. Create symbolic links in the new **tpl2** directory to these files:

   ```
   # ln -s
   abs_path_to/qls/p001/efp/2048k/in_use/efp_journal_1_file_n
   ame.jrnl tpl2/
   ```

   d. Repeat the above step for each file copied in **tpl**.

   See the **note** in step 4 above if more than one partition and/or more than one EFP size is in use, and make the appropriate adjustments as described there if necessary.

7. Restore the correct ownership of the **qls** directory:

   ```
   # chown -R qpidd:qpidd /absolute_path_to/qls
   ```

8. Restore SELinux contexts for qls directory

   ```
   # restorecon -FvvR /abs_path_to/qls
   ```

The upgrade is now complete, The broker can now be started. To confirm this, it is suggested that the broker be started with elevated logging, which will cause it to print additional messages about the Linearstore recovery process.

If the broker is started on the command-line, use the option **--log-enable info+** for the first restart, otherwise change the broker configuration file to use this log level prior to starting the broker as a service.

Once it has been established that all queues were successfully recovered and that all expected messages have been recovered, the broker may be stopped and the log level returned to its previous or default settings

See Also:

- Section 3.3.6, “Persistence Options”

Report a bug

1.1.7. Configure the Firewall for Message Broker Traffic

Before installing and configuring the message broker, you must allow incoming connections on the port it will use. The default port for message broker (AMQP) traffic is **5672**.

To allow this the firewall must be altered to allow network traffic on the required port. All steps must be run while logged in to the server as the **root** user.

Procedure 1.3. Configuring the firewall for message broker traffic

1. Open the **/etc/sysconfig/iptables** file in a text editor.
2. Add an **INPUT** rule allowing incoming connections on port **5672** to the file. The new rule must appear before any **INPUT** rules that **REJECT** traffic.

```
-A INPUT -p tcp -m tcp --dport 5672  -j ACCEPT
```

3. Save the changes to the **/etc/sysconfig/iptables** file.
4. Restart the **iptables** service for the firewall changes to take effect.

```
# service iptables restart
```

The firewall is now configured to allow incoming connections to the MariaDB database service on port **5672**.

Report a bug

1.2. Memory Requirements and Limitations

1.2.1. Memory Allocation Limit (32-bit)

A Broker running on a 32-bit operating system has a 3GB memory allocation limit. You can create a queue with greater than 3GB capacity on such a system, however when the queue reaches 3GB of queued messages, an attempt to send more messages to the queue will result in a memory allocation failure.

Report a bug

1.2.2. Impact of Transactions on the Journal

When transactions are used, a transaction ID (XID) is added to each record. The size of the XID is 24 bytes for local transactions. For distributed transactions, the user supplies the XID, which is usually obtained from the transaction monitor, and may be any size. In a transaction, in addition to message enqueue records, journal records are maintained for each message dequeue, and for each transaction abort or commit.

Report a bug

1.2.3. Messaging Broker Memory Requirements

The amount of memory required by a Broker is a function of the number and size of messages it will process simultaneously.

The size of a message is the combination of the size of the message header and the size of the message body.

Calculate message size

Note: Transactions increase the size of messages.

Procedure 1.4. Estimate message size

This method allows you to calculate message size theoretically.

1. Default message header content (such as Java timestamp and message-id): 55 bytes
2. Routing Key (for example: a routing key of "testQ" = 5 bytes)
3. Java clients add:
   - **`content-type`** (for "text/plain" it is 10 bytes)
   - **`user-id`** (user name passed to SASL for authentication, number of bytes equal to string size)
4. Application headers:
   - Application header overhead: 8 bytes
   - For any textual header properties: **`property_name_size + property_value_size + 4`** bytes

For example, sending a message using **`spout`** such as the following:

```
./run_example.sh org.apache.qpid.example.Spout -c=1 -
b="guest:guest@localhost:5672" -P=property1=value1 -P=property2=value2
"testQ; {create:always}" "123456789"
```

sends an AMQP message with message body size 9, while the message header will be of size:

- 55 bytes for the default size
- 5 bytes for the routing key "testQ"
- 10 bytes for content-type "text/plain"
- 5 bytes for user-id "guest"
- 8 bytes for using application headers
- 9+6+4 bytes for the first property
- 9+6+4 bytes for the second property
- Total header size: 121 bytes
- Total message size: 130 bytes

Procedure 1.5. Determine message size from logs

This method allows you to measure message sizes from a running broker.

1. Enable trace logging by adding the following to **`/etc/qpid/qpidd.conf`**:

   ```
   log-enable=trace+:qpid::SessionState::receiverRecord
   log-enable=info+
   log-to-file=/path/to/file.log
   ```

   Note that this logging will consume significant disk space, and should be turned off by removing these lines after the test is performed.

2. (Re)start the Broker.

3. Send a sample message pattern from a qpid client. This sample message pattern should correspond to your normal utilization, so that the message header and body average sizes match your projected real-world use case.

4. After the message pattern is sent, grep in the logfile for log records:

```
2012-10-16 08:56:20 trace guest@QPID.2fa0df51-6131-463e-90cc-
45895bea072c: recv cmd 2: header (121 bytes); properties=
{{MessageProperties: content-length=9; message-id=d096f253-56b9-33df-
9673-61c55dcba4ae; content-type=text/plain; user-id=guest;
application-headers=
{property1:V2:6:str16(value1),property2:V2:6:str16(value2)}; }
{DeliveryProperties: priority=4; delivery-mode=2;
timestamp=1350370580363; exchange=; routing-key=testQ; }}
```

   This example log entry contains both header size (121 bytes in this case) and message body size (9 bytes in this case, as content-length=9).

Message memory utilization on Broker

On the broker, memory is utilized to hold the message. In addition:

- A second instance of the message header is kept - one is stored as raw bytes, the other as a map.
- The Message object uses 600 bytes.
- Each message is guarded by three mutexes and monitor. These require 208 bytes.

So the total calculation of memory usage for a message is:

message_body + (message_header * 2) + 808 bytes

Using an average value for message body and header size, and multiplying this figure by the sum of queue depths will give you a saturated memory load figure for the Broker.

Note: an in-memory optimization for exchanges uses one copy of a message for all subscribed queues when the message is delivered to an exchange. This means that a broker with exchanges delivering to multiple queues will use significantly lower amounts of memory in normal operation. However, if the broker is restarted and the queued messages are read from disk, they are read *per-queue* and the full memory impact is experienced.

Report a bug

1.3. MRG 2 Features - Where Are They Now?

1.3.1. Configuration file changes

The MRG 2 configuration file was located at **`/etc/qpidd.conf`**. The configuration file for the MRG 3 broker is now located in **`/etc/qpid/qpidd.conf`**.

You can manually merge MRG 2 configuration into the new MRG 3 configuration file.

Report a bug

1.3.2. Cluster configuration changes

Clustering has changed, and MRG 2 clustering configuration should be removed from the configuration file. The following parameters must be removed:

- **`cluster-name`**
- **`cluster-mechanism`**
- **`cluster-url`**
- **`cluster-username`**
- **`cluster-password`**
- **`cluster-cman`**
- **`cluster-size`**
- **`cluster-clock-interval`**
- **`cluster-read-max`**

See Also:

- Section 9.1.1, “Changes to Clustering in MRG 3”

Report a bug

1.3.3. Flow-to-disk replacement

Flow-to-disk has been replaced by paged queues, so references to **`flow_to_disk`** must be removed from the configuration file.

See Also:

- Section 4.9.2, “Disk-paged Queues”

Report a bug

1.3.4. Linear Store

MRG 3 introduces a new *Linear Store* for persistent messages. The Linear Store allows the persistence journal to dynamically expand. This means that durable queues can be expanded without deleting and recreating them.

There is no way to upgrade from the MRG 2 journal. Changing to MRG 3 must be done from a clean store and all queues that use the journal must be redeclared.

See Also:

- Section 5.3, “Message Journal”

Report a bug

1.3.5. Address string and connection options

The new AMQP 1.0 qpid-jms client does not support the MRG-specific address strings used by the older AMQP 0.10 java client.

In the new client, the "queue name" and and "topic name" strings given to the new client represent only the name of the Queue or Topic to consume from or publish to.

Other concepts from the old client syntax such as subjects or operations to create and delete entities are not supported.

Report a bug

1.4. Application Migration

1.4.1. API support in MRG 3

Only **`qpid::types`** and **`qpid::messaging`** APIs are supported in MRG 3.

Applications that use other **`qpid::`** namespace APIs (for example, **`qpid::client`** [1] and **`qpid::types::Variant::fromString`** [2]) need to be rewritten to use only the supported APIs.

Report a bug

1.4.2. qpid::messaging Message::get/setContentObject()

Structured AMQP 1.0 messages can have the body of the message encoded in a variety of ways.

The Ruby and Python APIs do not decode the body of structured AMQP 1.0 message. A message sent as an AMQP 1.0 type can be received by these libraries, but the body is not decoded. Applications using the Ruby and Python APIs need to decode the body themselves.

The C++ and C# APIs have the new methods **`Message::getContentObject()`** and **`Message::setContentObject()`** to access the semantic content of structured AMQP 1.0 messages. These methods allow the body of the message to be accessed or manipulated as a Variant. Using these methods produces the most widely applicable code as they work for both protocol versions and work with map-, list-, text- or binary- messages.

The content object is a Variant, allowing the type to be determined, and also allowing the content to be automatically decoded.

The following C++ example demonstrates the new methods:

```
bool Formatter::isMapMsg(qpid::messaging::Message& msg) {
  return(msg.getContentObject().getType() == qpid::types::VAR_MAP);
}

bool Formatter::isListMsg(qpid::messaging::Message& msg) {
  return(msg.getContentObject().getType() == qpid::types::VAR_LIST);
}

qpid::types::Variant::Map Formatter::getMsgAsMap(qpid::messaging::Message&
msg) {
  qpid::types::Variant::Map intMap;
  intMap = msg.getContentObject().asMap();
  return(intMap);
}

qpid::types::Variant::List Formatter::getMsgAsList(qpid::messaging::Message&
msg) {
```

```
  qpid::types::Variant::List intList;
  intList = msg.getContentObject().asList();
  return(intList);
}
```

`Message::getContent()` and **`Message::setContent()`** continue to refer to the raw bytes of the content. The **`encode()`** and **`decode()`** methods in the API continue to decode map- and list- messages in the AMQP 0-10 format.

Report a bug

1.4.3. Ambiguous Addresses in AMQP 1.0

When a client uses an ambiguous address over AMQP 1.0 - where both a queue and an exchange of the same name exist and the client does not explicitly specify the node type - the *queue* is used by default. No exception or warning is printed out, and the client is not informed in any way of the ambiguity. A warning is logged by the broker when the node matches both a queue and an exchange and no specific type is specified.

When using an AMQP 0-10 client, in contrast, the following exception is reported: "**`Ambiguous address, please specify queue or topic as node type`**".

Report a bug

---

[1] https://bugzilla.redhat.com/show_bug.cgi?id=995039

[2] https://bugzilla.redhat.com/show_bug.cgi?id=1141230

Chapter 2. Start the Messaging Broker

2.1. Starting the Broker via command line vs as a service

When started as a service, the Broker reads its start-up options from a configuration file. When started at the command line, the Broker can read its start-up options either from a configuration file, or from command-line arguments.

Starting the Broker as a service is useful for production servers. The Broker can be configured to automatically start whenever the server is restarted. For development use, starting and re-starting the Broker with different configurations usually means that starting from the command line is useful.

Report a bug

2.2. Running the Broker at the command line

2.2.1. Start the Broker at the command line

Start the Broker

1. By default, the broker is installed in **`/usr/sbin/`**. If this is not on your path, you need to type the whole path to start the broker:

   ```
   /usr/sbin/qpidd -t
   ```

   You will see output similar to the following when the broker starts:

   ```
   [date] [time] info Loaded Module: libbdbstore.so.0
   [date] [time] info Locked data directory: /var/lib/qpidd
   [date] [time] info Management enabled
   [date] [time] info Listening on port 5672
   ```

   The **`-t`** or **`--trace`** option enables debug tracing, printing messages to the terminal.

   Note: The locked data directory **`/var/lib/qpidd`** is used for persistence, which is enabled by default.

Report a bug

2.2.2. Stop the Broker when started at the command line

1. To stop the broker, type **`CTRL`**+ **`C`** at the shell prompt

   ```
   [date] [time] notice Shutting down.
   [date] [time] info Unlocked data directory: /var/lib/qpidd
   ```

Live preview.

Report a bug

2.3. Running the Broker as a service

2.3.1. Run the Broker as a service

- In production scenarios, Red Hat Enterprise Messaging is usually run as a service. To start the broker as a service, with root privileges run the command:

```
service qpidd start
```

  The message broker starts with the message:

```
Starting Qpid AMQP daemon:                                 [  OK  ]
```

Report a bug

2.3.2. Stop the Broker service

- To check the status of a broker running as a service use the **`service status`** command. Stop the broker with the command **`service stop`**.

```
# service qpidd status
qpidd (pid PID) is running...

# service qpidd stop
Stopping Qpid AMQP daemon:                                 [  OK  ]
```

Report a bug

2.3.3. Configure the Broker service to start automatically when the server is started

On a production server the message broker typically starts automatically whenever the machine is restarted. To do this, you need to enable the **`qpidd`** service.

- Run the following command as root:

```
chkconfig qpidd on
```

The message broker qpidd demon will now start automatically when the server is started.

Report a bug

2.4. Run multiple Brokers on one machine

2.4.1. Running multiple Brokers

In a development environment it is possible to run multiple brokers on the same machine for testing and prototyping.

Report a bug

2.4.2. Start multiple Brokers

To run more than one broker on a single machine, each broker must run on different ports and use different directories for the journals.

1. Select two available ports, for example *5555* and *5556*.
2. Start each new broker, using the **`--data-dir`** command to specify a new data directory for each:

```
$ qpidd -p 5555 --data-dir /tmp/qpid/store/1
```

```
$ qpidd -p 5556 --data-dir /tmp/qpid/store/2
```

Report a bug

Chapter 3. Give Yourself (Broker) Options

3.1. Set Broker options at the command line

To set options for a single instance, add the option to the command line when you start the broker.

- This example uses the command line option **-t** to start the broker with debug tracing.

```
$ /usr/sbin/qpidd -t
```

Command-line options need to be provided each time the broker is invoked from the command line.

Report a bug

3.2. Set Broker options in a configuration file

To set the Broker options when running the Broker as a service, or to create a set of options that can be used when starting the Broker from the command-line, but without having to specify them each time on the command-line, use a configuration file.

1. Become the root user, and open the **/etc/qpid/qpidd.conf** file in a text editor.
2. This example uses the configuration file to enable debug tracing. Changes will take effect from the next time the broker is started and will be used in every subsequent session.

```
# Configuration file for qpidd
trace=1
```

3. If you are running the broker as a service, you need to restart the service to reload the configuration options.

```
# service qpidd restart
Stopping qpidd daemon:                     [  OK  ]
Starting qpidd daemon:                     [  OK  ]
```

4. If you are running the broker from the command-line, start the broker with no command-line options to use the configuration file.

```
# /usr/sbin/qpidd
[date] [time] info Locked data directory: /var/lib/qpidd
[date] [time] info Management enabled
[date] [time] info Listening on port 5672
```

Report a bug

3.3. Broker options

3.3.1. Options for running the Broker as a Daemon

Changes

- New in MRG 3.

Options for running the broker as a daemon	
-d	Run in the background as a daemon. Log messages from the broker are sent to **syslog** (**/var/log/messages**) by default.
-q	Shut down the broker that is currently running.
-c	Check if the daemon is already running. If it is running, return the process ID.
--wait=<*seconds*>	Wait <*seconds*> seconds during initialization and shutdown. If the daemon has not successfully completed initialization or shutdown within this time, an error is returned. On shutdown, the daemon will wait this period of time to allow the broker to shutdown before reporting success or failure. This option must be used in conjunction with the **-d** option, or it will be ignored.

Report a bug

3.3.2. General Broker options

The broker has a number of general command-line options. There are additional High Availability options that are described in Section 9.1.6, “Broker HA Options”.

List of General Broker Command-line Options

-h

Displays the help message.

--interface <*ipaddr*>

Listen on the specified network interface. Can be used multiple times for multiple network interfaces. This option supports IPv4 and IPv6 addresses. You can use an explicit address, or the name of a network adapter (for example *eth0* or *em1*). If you specify a network adapter name the broker will bind to all addresses bound to that adapter.

--link-heartbeat-interval <*seconds*>

The number of seconds to wait for a federated link heart beat. By default this is 120 seconds.

--link-maintenance-interval <*seconds*>

The number of seconds to wait for backup brokers to verify the link to the primary, and reconnect if required. This value defaults to 2.

-p <*Port_Number*>

Instructs the broker to use the specified port. Defaults to port 5672. It is possible to run multiple brokers simultaneously by using different port numbers.

--paging-dir <*directory*>

Directory to use for disk-paged queues.

--socket-fd <*fd*>

Use an existing socket specified by its file descriptor. Can be used multiple times for multiple sockets. This is useful when the broker is started by a parent process, for example during testing.

-t

This option enables verbose log messages, for debugging only.

--tcp-nodelay *on|off*

Disable ack on TCP. This increases throughput, especially in synchronous operations. This is set to **on** by default. You can set this in the configuration file using **`QPID_TCP_NODELAY=on|off`**

-v

Displays the installed version.

Report a bug

3.3.3. Logging

By default, log output is sent to **`stderr`** if the broker is run on the command line, or to **`syslog`** (**`/var/log/messages/`**), if the broker is run as a service.

Table 3.1. Logging Options

Options for logging with syslog	
`-t [--trace]`	Enables all logging
`--log-disable RULE`	Disables logging for selected levels and components (opt-out). *RULE* is in the form **`LEVEL[+]:[:PATTERN]`**. Levels are one of: **`trace, debug, info, notice, warning, error, critical.`**. This allows uninteresting log messages to be dropped during debugging. This can be used multiple times.
`--log-enable RULE (notice+)`	Enables logging for selected levels and components. RULE is in the form **`LEVEL[+]:[:PATTERN]`**. Levels are one of: **`trace, debug, info, notice, warning, error, critical.`**. For example: **`--log-enable warning+`** logs all warning, error, and critical messages. **`--log-enable debug:framing`** logs debug messages from the framing namespace. This can be used multiple times.
`--log-time yes|no`	Include time in log messages
`--log-level yes|no`	Include severity level in log messages
`--log-source`	Include source file:line in log messages
`--log-thread yes|no`	Include thread ID in messages
`--log-function yes|no`	Include function signature in log messages
`--log-hires-timestamp yes|no (0)`	Use hi-resolution timestamps in log messages
`--log-category yes|no (1)`	Include category in log messages
`--log-prefix STRING`	Prefix to append to all log messages
`--log-to-stderr yes|no`	Send logging output to **`stderr`**. Enabled by default when run from command line.
`--log-to-stdout yes|no`	Send logging output to **`stdout`**.
`--log-to-file FILE`	Send log output to the specified filename. *FILE*.

Options for logging with syslog	
--log-to-syslog yes|no	Send logging output to **syslog**. Enabled by default when run as a service.
--syslog-name *NAME*	Specify the name to use in **syslog** messages. The default is **qpidd**.
--syslog-facility *LOG_XXX*	Specify the facility to use in **syslog** messages. The default is **LOG_DAEMON**.

See Also:

- Chapter 7, *Logging*

Report a bug

3.3.4. Modules

Table 3.2. Options for using modules with the broker

Options for using modules with the broker	
--load-module *MODULENAME*	Use the specified module as a plug-in.
--module-dir <DIRECTORY>	Use a different module directory.
--no-module-dir	Ignore module directories.

Getting Help with Modules

To see the help text for modules, use the **--help** command:

```
# /usr/sbin/qpidd --help
```

Report a bug

3.3.5. Default Modules

The following modules are installed and are loaded by default:

- XML exchange type
- Persistence
- Clustering

Report a bug

3.3.6. Persistence Options

Table 3.3. Journal Options

Option	Default	Description

Option	Default	Description
--store-dir *DIR*	See the description for more information.	Store directory location for persistence journals. The default is **/var/lib/qpidd** when run as a daemon, or **~/.qpidd** when run from the command line. This option can be used to override the default location, or the location specified by **--data-dir**. It is required if **--no-data-dir** is used.
--truncate *yes*|*no*	no	If *yes*|*true*|*1*, will truncate the store (discard any existing records). If *no*|*false*|*0*, will preserve the existing store files for recovery.
--wcache-page-size *N*	32	Size of the pages in the write page cache in KiB. Allowable values - powers of two, starting at 4: 4, 8, 16, 32... Lower values decrease latency at the expense of throughput.
--wcache-num-pages *N*	16	Number of pages in the write page cache. Minimum value: 4.
--tpl-wcache-page-size *N*	4	Size of the pages in the transaction prepared list write page cache in KiB. Allowable values - powers of two, starting at 4: 4, 8, 16, 32... Lower values decrease latency at the expense of throughput.
--tpl-wcache-num-pages *N*	16	Number of pages in the transaction prepared list write page cache. Minimum value: 4.

Option	Default	Description
--efp-partition *N*	1	Empty File Pool broker partition to use for finding empty journal files. If this option is not specified, the default partition value of 1 is used. This value translates to the broker partition **p001**). To select a partition and journal file size other than the broker default, use **qpid-config** and the **--efp-partition** and **--efp-file-size** options to select another partition and/or size combination. For example: **qpid-config add queue test-queue-5 --durable --efp-partition 5 --efp-file-size 8192**  **Important** The partition must exist prior to starting the broker.
--efp-file-size *N*	2048	Empty File Pool broker journal file size in KiB. Must be a multiple of 4 KiB. If this option is not specified, the default file size of 2048 KiB is used. To use the option, see the command example in **--efp-partition**.

See Also:

- Section 1.1.5, “Linearstore Custom Broker EFP Partitions”

Report a bug

3.3.7. Queue Options

Table 3.4. Queue Options

Option	Default	Description

Option	Default	Description
--queue-purge-interval	600	Specifies the time in seconds that the broker browses all queues and purges all messages with an expired Time To Live (TTL). Use this option for queues where consumers are consistently behind producers in message processing to ensure expired messages are not held past their TTL.

Report a bug

3.3.8. Resource Quota Options

The maximum number of connections can be restricted with the **--max-connections** broker option.

Table 3.5. Resource Quota Options

Option	Description	Default Value
--max-connections *N*	Total concurrent connections to the broker.	500
--max-negotiate-time *N*	The time during which initial protocol negotiation must succeed. This prevents resource starvation by badly behaved clients or transient network issues that prevent connections from completing.	500
--session-max-unacked *N*	The broker will send messages on a session without waiting for acknowledgement up to this limit (or sooner, if the aggregate link credit for the session is lower). When this limit is reached, the broker will wait for acknowledgement from the client before sending more messages.	5000 (or approxiamately 625 KB / session)

Notes

- **--max-connections** is a qpid core limit and is enforced whether ACL is enabled or not.
- **--max-connections** is enforced per Broker. In a cluster of *N* nodes where all Brokers set the maximum connections to 20 the total number of allowed connections for the cluster will be *N**20.
- **--session-max-unacked** helps control memory use in cases where a large number of sessions are used with AMQP 1.0, which allocates a per-session buffer for unacknowledged message deliveries.

- **`--session-max-unacked`** can be used to make each session's buffer smaller, if the broker has a large number of sessions and memory overhead is an issue.

ACL-based Quotas

To enable ACL-based quotas, an ACL file must be loaded:

Table 3.6. ACL Command-line Option

Option	Description	Default Value
`--acl-file` ***`FILE (policy.acl)`***	The policy file to load from, loaded from data dir.	

When an ACL file is loaded, the following ACL options can be specified at the command-line to enforce resource quotas:

Table 3.7. ACL-based Resource Quota Options

Option	Description	Default Value
`--connection-limit-per-user` ***`N`***	The maximum number of connections allowed per user. 0 implies no limit.	0
`--connection-limit-per-ip` ***`N`***	The maximum number of connections allowed per host IP address. 0 implies no limit.	0
`--max-queues-per-user` ***`N`***	Total concurrent queues created by individual user	0

Notes

- In a cluster system the actual number of connections may exceed the connection quota value **`N`** by one less than the number of member nodes in the cluster. For example: in a 5-node cluster, with a limit of 20 connections, the actual number of connections can reach 24 before limiting takes place.
- Cluster connections are checked against the connection limit when they are established. The cluster connection is denied if a free connection is not available. After establishment, however, a cluster connection does not consume a connection.
- Allowed values for **`N`** are 0..65535.
- These limits are enforced per *cluster*.
- A value of zero (0) disables that option's limit checking.
- Per-user connections are identified by the authenticated user name.
- Per-ip connections are identified by the **`<broker-ip><broker-port>-<client-ip><client-port>`** tuple which is also the management connection index.
  - With this scheme host systems may be identified by several names such as **`localhost`** IPv4, **`127.0.0.1`** IPv4, or **`::1`** IPv6, and a separate set of connections is allowed for each name.
  - Per-IP connections are counted regardless of the user credentials provided with the connections. An individual user may be allowed 20 connections but if the client host has a 5 connection limit then that user may connect from that system only 5 times.

Report a bug

3.3.9. Security Options

Changes

- New for MRG 3.

Table 3.8. General Broker Options

Security options for running the broker	
--ssl-use-export-policy	Use NSS export policy
--ssl-cert-password-file *<PATH>*	Required. Plain-text file containing password to use for accessing certificate database.
--ssl-cert-name *<NAME>*	Name of the certificate to use. Default is **localhost.localdomain**.
--ssl-cert-db *<PATH>*	Required. Path to directory containing certificate database.
--ssl-port *<NUMBER>*	Port on which to listen for SSL connections. If no port is specified, port 5671 is used. If the SSL port chosen is the same as the port for non-SSL connections (i.e. if the --ssl-port and --port options are the same), both SSL encrypted and unencrypted connections can be established to the same port. However in this configuration there is no support for IPv6.
--ssl-require-client-authentication	Require SSL client authentication (i.e. verification of a client certificate) during the SSL handshake. This occurs before SASL authentication, and is independent of SASL. This option enables the EXTERNAL SASL mechanism for SSL connections. If the client chooses the EXTERNAL mechanism, the client's identity is taken from the validated SSL certificate, using the CN, and appending any DC's to create the domain. For instance, if the certificate contains the properties **CN=bob**, **DC=acme**, **DC=com**, the client's identity is **bob@acme.com**. If the client chooses a different SASL mechanism, the identity take from the client certificate will be replaced by that negotiated during the SASL handshake.
--ssl-sasl-no-dict	Do not accept SASL mechanisms that can be compromised by dictionary attacks. This prevents a weaker mechanism being selected instead of EXTERNAL, which is not vulnerable to dictionary attacks.
--require-encryption	This will cause qpidd to only accept encrypted connections. This means only clients with EXTERNAL SASL on the SSL-port, or with GSSAPI on the TCP port.

Security options for running the broker	
--listen-disable *PROTOCOL*	Disable connections over the specified protocol. For example: **--listen-disable tcp** disables connections over TCP and forces the broker to only accept connections on the SSL-port.

See Also:

- Section 8.2.3, “Enable SSL on the Broker”

Report a bug

3.3.10. Transactions Options

Table 3.9. Options for transactions

Option	Description
--dtx-default-timeout <*seconds*>	By default: 60 seconds. Journal records for DTX transactions are deleted after the specified number of seconds. This occurs when an external Transaction Manager (TM) prepares a DTX transaction but does not commit or abort it. After the specified number of seconds these are considered to be orphaned entries and are expunged.

Report a bug

Chapter 4. Queues

4.1. Message Queue

Message Queues are the mechanism for consuming applications to subscribe to messages that are of interest.

Queues receive messages from exchanges, and buffer those messages until they are consumed by message consumers. Those message consumers can browse the queue, or can acquire messages from the queue. Messages can be returned to the queue for redelivery, or they can be rejected by a consumer.

Multiple consumers can share a queue, or a queue may be exclusive to a single consumer.

Message producers can create and bind a queue to an exchange and make it available for consumers, or they can send to an exchange and leave it up to consumers to create queues and bind them to the exchange to receive messages of interest.

Temporary private message queues can be created and used as a response channel. Message queues can be set to be deleted by the broker when the application using them disconnects. They can be configured to group messages, to update messages in the queue with newly-arriving copies of messages, and to prioritize certain messages.

Another way of managing message queues, specifically in the area of message Time To Live (TTL), is to use the **`--queue-purge-interval`**. While this is not a qpid-config option, it is worth understanding that message TTL can be configured, and when the purge attempt is successful the messages are subsequently removed.

Refer to Section 3.3.7, “Queue Options” for details about this broker option.

Report a bug

4.2. Create and Configure Queues using qpid-config

The **`qpid-config`** command line tool can be used to create and configure queues.

The complete command reference for **`qpid-config`** is available by running the command with the **`--help`**switch:

```
qpid-config --help
```

When no server is specified, **`qpid-config`** runs against the message broker on the current machine. To interact with a message broker on another machine, use the **`-a`** or **`--broker-addr`** switch. For example:

```
qpid-config -a server2.testing.domain.com
```

The argument for the broker address option can specify a username, password, and port as well:

```
qpid-config -a user1/secretpassword@server2.testing.domain.com:5772
```

To create a queue, use the **`qpid-config add queue`** command. This command takes the name for the new queue as an argument, and [optionally] queue options.

A simple example, creating a queue called **`testqueue1`** on the message broker running on the local machine:

```
qpid-config add queue testqueue1
```

Here are the various options that you can specify when creating a queue with **qpid-config**:

Table 4.1. Options for qpid-config add queue

Options for qpid-config add queues	
--alternate-exchange *queue name*	Name of the alternate exchange. When the queue is deleted, all remaining messages in this queue are routed to this exchange. Messages rejected by a queue subscriber are also sent to the alternate exchange.
--durable	The new queue is durable. It will be recreated if the server is restarted, along with any undelivered messages marked as PERSISTENT sent to this queue.
--file-count *integer*	The number of files in the queue's persistence journal. Up to a maximum of 64. Attempts to specify more than 64 result in the creation of 64 journal files.
--file-size *integer*	File size in pages (64KiB/page).
--max-queue-size *integer*	Maximum in-memory queue size as bytes. Note that on 32-bit systems queues will not go over 3GB, regardless of the declared size.
--max-queue-count *integer*	Maximum in-memory queue size as a number of messages.
--limit-policy *[none, reject, ring]*	Action to take when queue limit is reached.
--flow-stop-size *integer*	Turn on sender flow control when the number of queued bytes exceeds this value.
--flow-resume-size *integer*	Turn off sender flow control when the number of queued bytes drops below this value.
--flow-stop-count *integer*	Turn on sender flow control when the number of queued messages exceeds this value.
--flow-resume-count	Turn off sender flow control when the number of queued messages drops below this value.
--group-header	Enable message groups. Specify name of header that holds group identifier.
--shared-groups	Allow message group consumption across multiple consumers.
--argument *name=value*	Specify a key-value pair to add to the queue arguments. This can be used, for example, to specify **no-local=true** to suppress loopback delivery of self-generated messages.

Note that you cannot create an exclusive queue using **qpid-config**, as an exclusive queue is only available in the session where it is created.

See Also:

- Section 12.1.2, “Using qpid-config”
- Section 4.6.2, “Declaring a Last Value Queue”
- Section 4.5, “Ignore Locally Published Messages”

- Appendix A, *Exchange and Queue Declaration Arguments*

Report a bug

4.3. Memory Allocation Limit (32-bit)

A Broker running on a 32-bit operating system has a 3GB memory allocation limit. You can create a queue with greater than 3GB capacity on such a system, however when the queue reaches 3GB of queued messages, an attempt to send more messages to the queue will result in a memory allocation failure.

Report a bug

4.4. Exclusive Queues

Exclusive queues can only be used in one session at a time. When a queue is declared with the exclusive property set, that queue is not available for use in any other session until the session that declared the queue has been closed.

If the server receives a declare, bind, delete or subscribe request for a queue that has been declared as exclusive, an exception will be raised and the requesting session will be ended.

Note that a session close is not detected immediately. If clients enable heartbeats, then session closes will be determined within a guaranteed time. See the client APIs for details on how to set heartbeats in a given API.

Report a bug

4.5. Ignore Locally Published Messages

You can configure a queue to discard all messages published using the same connection as the session that owns the queue. This suppresses a message loop-back when an application publishes messages to an exchange that it is also subscribed to.

To configure a queue to ignore locally published messages, use the **`no-local`** key in the queue declaration as a key:value pair. The value of the key is ignored; the presence of the key is sufficient.

For example, to create a queue that discards locally published messages using **`qpid-config`**:

```
qpid-config add queue noloopbackqueue1 --argument no-local=true
```

Note that multiple distinct sessions can share the same connection. A queue set to ignore locally published messages will ignore all messages from the *connection* that declared the queue, so all sessions using that connection are local in this context.

Report a bug

4.6. Last Value (LV) Queues

4.6.1. Last Value Queues

Last Value Queues allow messages in the queue to be overwritten with updated versions. Messages sent to a Last Value Queue use a header key to identify themselves as a version of a message. New messages with a matching key value arriving on the queue cause any earlier message with that key to be discarded. The result is that message consumers who browse the queue receive the latest version of a message only.

Report a bug

4.6.2. Declaring a Last Value Queue

Last Value Queues are created by supplying a **qpid.last_value_queue_key** when creating the queue.

For example, to create a last value queue called **stock-ticker** that uses **stock-symbol** as the key, using **qpid-config**:

```
qpid-config add queue stock-ticker --argument
qpid.last_value_queue_key=stock-symbol
```

To create the same queue in an application:

Python

```
myLastValueQueue = mySession.sender("stock-ticker;{create:always,
node:{type:queue, x-declare:{arguments:{'qpid.last_value_queue_key':
'stock-symbol'}}}}")
```

Both string and integer values can be provide as the last value. Using the example queue created above, valid values for the *stock-symbol* key would include "**RHT**", "**JAVA**", and other string values; and also **3**, **15**, and other integer values.

Report a bug

4.7. Message Groups

4.7.1. Message Groups

Message Groups allow a sender to indicate that a group of messages should all be handled by the same consumer. The sender sets the header of messages to identify them as part of the same group, then sends the messages to a queue that has message grouping enabled.

The broker ensures that a single consumer gets exclusive access to the messages in a group, and that the messages in a group are delivered and re-delivered in the order they were received.

Note that Message Grouping cannot be used in conjunction with Last Value Queue or Priority Queuing.

The implementation of Message Groups is described in a specification attached to its feature request: QPID-3346: Support message grouping with strict sequence consumption across multiple consumers.

Report a bug

4.7.2. Message Group Consumer Requirements

The correct handling of group messages is the responsibility of both the broker and the consumer. When a consumer fetches a message that is part of a group, the broker makes that consumer the owner of that message group. All of the messages in that group will be visible only to that consumer *until the consumer acknowledges receipt of all the messages it has fetched from that group*. When the consumer acknowledges all the messages it has fetched from the group, the broker releases its ownership of the group.

The consumer should acknowledge all of the fetched messages in the group at once. The purpose of message grouping is to ensure that all the messages in the group are dealt with by the same consumer. If a consumer takes grouped messages from the queue, acknowledges some of them and then disconnects due

to a failure, the unacknowledged messages in the group will be released and become available to other consumers. However, the acknowledged messages in the group have been removed from the queue, so now part of the group is available on the queue with the header **`redelivered=True`**, and the rest of the group is missing.

For this reason, consuming applications should be careful to acknowledge all grouped messages at once.

Report a bug

4.7.3. Configure a Queue for Message Groups using qpid-config

This example **`qpid-config`** command creates a queue called "MyMsgQueue", with message grouping enabled and using the header key "GROUP_KEY" to identify message groups.

```
qpid-config add queue MyMsgQueue --group-header="GROUP_KEY" --shared-groups
```

Report a bug

4.7.4. Default Group

All messages arriving to a queue with message groups enabled with no group identifier in their header are considered to belong to the same "default" group. This group is **`qpid.no-group`**. If a message cannot be assigned to any other group, it is assigned to this group.

Report a bug

4.7.5. Override the Default Group Name

When a queue has message groups enabled, messages are grouped based on a match with a header field. Messages that have no match in their headers for a group are assigned to the default group. The default group is preconfigured as **`qpid.no-group`**. You can change this default group name by supplying a value for the **`default-message-group`** configuration parameter to the broker at start-up. For example, using the command line:

```
qpidd --default-message-group "EMPTY-GROUP"
```

Report a bug

4.8. Alternate Exchanges

4.8.1. Rejected and Orphaned Messages

Messages can be explicitly *rejected* by a consumer. When a message is fetched over a reliable link, the consumer must acknowledge the message for the broker to release it. Instead of acknowledging a message, the consumer can *reject* the message. The broker discards rejected messages, unless an alternate exchange has been specified for the queue, in which case the broker routes rejected messages to the alternate exchange.

Messages are orphaned when they are in a queue that is deleted. Orphaned messages are discarded, unless an alternate exchange is configured for the queue, in which case they are routed to the alternate exchange.

Report a bug

4.8.2. Alternate Exchange

An *alternate exchange* provides a delivery alternative for messages that cannot be delivered via their initial routing.

For an alternate exchange specified for a queue, two types of unroutable messages are sent to the alternate exchange:

1. Messages that are acquired and then rejected by a message consumer (*rejected messages*).
2. Unacknowledged messages in a queue that is deleted (*orphaned messages*).

For an alternate exchange specified for an exchange, one type of unroutable messages is sent to the alternate exchange:

1. Messages sent to the exchange with a routing key for which there is no matching binding on the exchange.

Note that a message will not be re-routed a second time to an alternate exchange if it is orphaned or rejected after previously being routed to an alternate exchange. This prevents the possibility of an infinite loop of re-routing.

However, if a message is routed to an alternate exchange and is unable to be delivered by that exchange because there is no matching binding, then it *will* be re-routed to that exchange's alternate exchange, if one is configured. This ensures that fail-over to a dead letter queue is possible.

Report a bug

4.9. Queue Sizing

4.9.1. Controlling Queue Size

Controlling the size of queues is an important part of performance management in a messaging system.

When queues are created, you can specify a maximum queue size (**`qpid.max_size`**) and maximum message count (**`qpid.max_count`**) for the queue.

`qpid.max_size` is specified in bytes. **`qpid.max_count`** is specified as the number of messages.

The following **`qpid-config`** creates a queue with a maximum size in memory of 200MB, and a maximum number of 5000 messages:

```
qpid-config add queue my-queue --max-queue-size=204800000 --max-queue-count
5000
```

In an application, the **`qpid.max_count`** and **`qpid.max_size`** directives go inside the **`arguments`** of the **`x-declare`** of the **`node`**. For example, the following address will create the queue as the **`qpid-config`** command above:

Python

```
tx = ssn.sender("my-queue; {create: always, node: {x-declare: {'auto-
delete': True, arguments:{'qpid.max_count': 5000, 'qpid.max_size':
204800000}}}}")
```

Note that the **qpid.max_count** attribute will only be applied if the queue does not exist when this code is executed.

Behavior when limits are reached: qpid.policy_type

The behavior when a queue reaches these limits is configurable. By default, on non-**durable** queues the behavior is **reject**: further attempts to send to the queue result in a **TargetCapacityExceeded** exception being thrown at the sender.

The configurable behavior is set using the **qpid.policy_type** option. The possible values are:

reject

Message publishers throw an exception **TargetCapacityExceeded**. This is the default behavior for non-**durable** queues.

ring

The oldest messages are removed to make room for newer messages.

The following example **qpid-config** command sets the limit policy to **ring**:

```
qpid-config add queue my-queue --max-queue-size=204800 --max-queue-count
5000 --limit-policy ring
```

The same thing is achieved in an application like so:

Python

```
tx = ssn.sender("my-queue; {create: always, node: {x-declare: {'auto-
delete': True, arguments:{'qpid.max_count': 5000, 'qpid.max_size':
204800, 'qpid.policy_type': 'ring'}}}}")
```

See Also:

- Section 4.11, “Producer Flow Control”
- Section 4.9.3, “Detect Overwritten Messages in Ring Queues”
- Section 4.9.2, “Disk-paged Queues”

Report a bug

4.9.2. Disk-paged Queues

MRG 3 replaces the MRG 2 **flow-to-disk** queue policy with more performant *paged queues*. Paged queues are backed by a file, and a configurable number of pages of messages are held in memory. Paged queues balance responsive performance (by holding messages in-memory and writing pages of messages rather than individual messages to disk) with load capacity (by allowing the queue to use the file system for additional storage).

Messages are stored in *pages*. The size of the memory page is configurable, and should be set to greater than the largest anticipated message size, to allow the message to fit inside a page. Messages larger than the page size are rejected by the broker.

The number of pages to hold in memory is configurable. When the configured maximum number of pages of messages in-memory are filled, further messages cause a page to be swapped out of memory to the disk file

to allow an empty in-memory page to receive further messages. Pages are proactively written to disk while still held in memory to optimize performance. When a message is requested from a page that is not in-memory, that page is retrieved from the disk. If the in-memory page limit has been reached, then a page in-memory is sent to disk to allow the new page to be loaded.

The upper limit of number of pages supported by the broker is determined by the system mapping limit. This is a kernel attribute and can be examined by **`cat /proc/sys/vm/max_map_count`**, and set at run-time with:

```
echo 100000 >/proc/sys/vm/max_map_count
```

Note that this method of setting the limit is non-permanent and is erased after a restart. To configure the limit in a persistent way, use the **`/etc/sysctl.conf`** file.

The size of a paged queue can be controlled by the usual size and message count limits.

Note that the disk-based storage for paged queues is not inherently persistent. It is used at run-time to manage the queue and balance memory use, and is not persistent across broker restarts. Paged queues can be declared **`durable`**, which provides persistence to messages that request it.

Limitations of Paged Queues

- A paged queue cannot handle a message larger than the page size, so the queue must be configured with pages at least as big as the largest anticipated message.
- A paged queue cannot also be a LVQ or Priority queue. An exception is thrown by an attempt to create a paged queue with LVQ or Priority specified.

Creating a Paged Queue

To configure a queue as a paged queue, specify the argument **`qpid.paging`** as **`true`** when declaring the queue.

The additional configuration options are:

`qpid.max_pages_loaded`

Controls how many pages are allowed to be held in memory at any given time. Default value is 4.

`qpid.page_factor`

Controls the size of the page. Default value is 1. The value is a multiple of the platform-defined page size. On Linux the platform-defined page size can be examined using the command **`getconf PAGESIZE`**. It is typically 4k, depending on your CPU architecture.

Example

The following command line example demonstrates creation of a paged queue:

```
qpid-config add queue my-paged-queue --argument qpid.paging=True --argument
qpid.max_pages_loaded=100 --argument qpid.page_factor=1
```

The same is accomplished in code in the following manner:

Python

```
tx = session.sender("my-paged-queue; {create: always, node: {x-
declare: {'auto-delete': True, arguments:{'qpid.page_factor': 1,
'qpid.max_pages_loaded': 100, 'qpid.paging': True}}}}")
```

Report a bug

4.9.3. Detect Overwritten Messages in Ring Queues

Ring queues overwrite older messages with incoming messages when the queue capacity is reached. Some applications need to be aware when messages have been overwritten. This can be achieved by declaring the queue with the **qpid.queue_msg_sequence** argument.

The **qpid.queue_msg_sequence** argument accepts a single string value as its parameter. This string value is added by the broker as a message property on each message that comes through the ring queue, and the property is set to a sequentially incrementing integer value.

Applications can examine the value of the **qpid.queue_msg_sequence** specified property on each message to determine if interim messages have been overwritten in the ring queue, and response appropriately.

Note that the message sequence must be examined by a stateful application to detect a break in the sequence. An exclusive queue with a single consumer is able to do this. If multiple consumers take messages from the queue the message sequence is split between consumers and they have no way to tell if a message has been overwritten.

Note also that the message sequence is not persisted, even with persistent messages sent to a durable queue, so a broker restart causes sequence discontinuity.

The following code demonstrates the use of **qpid.queue_msg_sequence**:

Python

```
import sys
from qpid.messaging import *
from qpid.datatypes import Serial

conn = Connection.establish("localhost:5672")
ssn = conn.session()

name="ring-sequence-queue"
key="my_sequence_key"
addr = "%s; {create:sender, delete:always, node: {x-declare:
{arguments: {'qpid.queue_msg_sequence':'%s',
'qpid.policy_type':'ring', 'qpid.max_count':4}}}}"  % (name, key)
sender = ssn.sender(addr)

msg = Message()
sender.send(msg)

receiver = ssn.receiver(name)
msg = receiver.fetch(1)

try:
    seqNo = Serial(long(msg.properties[key]))
    if seqNo != 1:
        print "Unexpected sequence number. Should be 1. Received
```

```
(%s)" % seqNo
    else:
       print "Received message with sequence number 1"
except:
    print "Unable to get key (%s) from message properties" % key

"""
Test that sequence number for ring queues shows gaps when queue
messages are overwritten
"""

msg = Message()
sender.send(msg)
msg = receiver.fetch(1)
seqNo = Serial(long(msg.properties[key]))

print "Received second message with sequence number %s" % seqNo
# send 5 more messages to overflow the queue
for i in range(5):
    sender.send(msg)

msg = receiver.fetch(1)
seqNo = msg.properties[key]
if seqNo != 3:
    print "Unexpected sequence number. Should be 3. Received (%s) -
Message overwritten in ring queue." % seqNo
receiver.close()
ssn.close()
```

The message sequence number is transferred as an unsigned 32 bit integer, so it wraps around at 2^32. In Python, use the **`Serial`** class from **`qpid.datatype`** to handle the wrapping.

Report a bug

4.9.4. Enforcing Queue Size Limits via ACL

The maximum queue size can be enforced via an ACL. This allows the administrator to disallow users from creating queues that could consume too many system resources.

CREATE QUEUE rules have ACL rules that limit the upper and lower bounds of both in-memory queue and on-disk queue store sizes.

Table 4.2. Queue Size ACL Rules

User Option	ACL Limit Property	Units
`qpid.max_size`	queuemaxsizelowerlimit	bytes
	queuemaxsizeupperlimit	bytes
`qpid.max_count`	queuemaxcountlowerlimit	messages
	queuemaxcountupperlimit	messages
`qpid.max_pages_loaded`	pageslowerlimit	pages
	pagesupperlimit	pages
`qpid.page_factor`	pagefactorlowerlimit	integer (multiple of the platform-defined page size)

User Option	ACL Limit Property	Units
	pagefactorupperlimit	integer (multiple of the platform-defined page size)

ACL Limit Properties are evaluated when the user presents one of the options in a CREATE QUEUE request. If the user's option is not within the limit properties for an ACL Rule that would allow the request, then the rule is matched with a Deny result.

Limit properties are ignored for Deny rules.

Example:

```
# Example of ACL specifying queue size constraints
# Note: for legibility this acl line has been split into multiple lines.
acl allow bob@QPID create queue name=q6 queuemaxsizelowerlimit=500000
                                        queuemaxsizeupperlimit=1000000
                                        queuemaxcountlowerlimit=200
                                        queuemaxcountupperlimit=300
```

These limits come into play when a queue is created as illustrated here:

C++

```
int main(int argc, char** argv) {
const char* url = argc>1 ? argv[1] : "amqp:tcp:127.0.0.1:5672";
const char* address = argc>2 ? argv[2] :
    "message_queue; “
    “ { create: always, “
    “   node: “
    “   { type: queue, “
    “     x-declare: ”
    “     { arguments: “
    “       { qpid.max_count:101,”
    “         qpid.max_size:1000000”
    “       }”
    “     }”
    “   }”
    “ }";
std::string connectionOptions = argc > 3 ? argv[3] : "";

Connection connection(url, connectionOptions);
try {
    connection.open();
    Session session = connection.createSession();
    Sender sender = session.createSender(address);
...
```

This queue can also be created with the **`qpid-config`** command:

```
qpid-config add queue --max-queue-size=1000000 --max-queue-count=101
```

When the ACL rule is processed assume that the actor, action, object, and object name all match and so this allow rule matches for the allow or deny decision. However, the ACL rule is further constrained to limit 500000 <= max_size <= 1000000 and 200 <= max_count <= 300. Since the **`queue_option max_count`** is 101 then the size limit is violated (it is too low) and the allow rule is returned with a deny decision.

Note that it is not mandatory to set *both* an upper limit *and* a lower limit. It is possible to set only a lower limit, or only an upper limit.

Report a bug

4.9.5. Queue Threshold Alerts (Edge-triggered)

In MRG 3, Queue Threshold alerts are *edge-triggered*. Thresholds are configured per-queue and are:

- **`qpid.alert_count_up`** - upper threshold (messages)
- **`qpid.alert_size_up`** - upper threshold (bytes)
- **`qpid.alert_count_down`** - lower threshold (messages)
- **`qpid.alert_size_down`** - lower threshold (bytes)

By default, the upward threshold is set to the global threshold ratio multiplied by the maximum size/count of the queue. The global threshold ratio can be specified using the **`--default-event-threshold-ratio`** command line option, otherwise it defaults to 80%.

By default, the downward threshold is set to one half of the default upward threshold.

Note: The upper and lower thresholds should have a gap between them to limit event rates.

Events

There are two different events:

Threshold crossed increasing

The increasing event is raised when the queue depth goes from (***`upper-threshold - 1`***) to ***`upper-threshold`*** and the increasing event flag is not already set. When an increasing event occurs the increasing event flag is set. The increasing event flag must be cleared (by a decreasing event) before further increasing events will be raised. This prevents multiple retriggering of this event by fluctuation of queue depth around the upper-threshold.

Threshold crossed decreasing

The decreasing event is raised when the increasing event flag is set and the queue depth goes from (***`lower-threshold + 1`***) to ***`lower-threshold`***. The decreasing event clears the increasing event flag, allowing further increasing events to be triggered and preventing multiple retriggering of this event by fluctuation of queue depth around the lower-threshold.

The events are sent via the QMF framework. You can subscribe to the event messages by listening to the addresses:

- **`qmf.default.topic/agent.ind.event.org_apache_qpid_broker.queueThresholdCrossedUpward.#`**
- **`qmf.default.topic/agent.ind.event.org_apache_qpid_broker.queueThresholdCrossedDownward.#`**

Events are sent as map messages:

qmf::org::apache::qpid::broker::EventQueueThresholdCrossedUpward(name, count, size)

qmf::org::apache::qpid::broker::EventQueueThresholdCrossedDownward(name, count, size)

The following code demonstrates subscribing to and consuming threshold event messages:

Window One

Python

```
import sys
from qpid.messaging import *
conn = Connection.establish("localhost:5672")
session = conn.session()
rcv =
session.receiver("qmf.default.topic/agent.ind.event.org_apache_qpid_b
roker.queueThresholdCrossedUpward.#")
while True:
   event = rcv.fetch()
   print "Threshold exceeded on queue %s" % event.content[0]
["_values"]["qName"]
   print "at a depth of %s messages, %s bytes" % (event.content[0]
["_values"]["msgDepth"], event.content[0]["_values"]["byteDepth"])
   session.acknowledge()
```

Window Two

Python

```
import sys
from qpid.messaging import *
connection = Connection.establish("localhost:5672")
session = connection.session()
rcv = session.receiver("threshold-queue; {create:always, node:{x-
declare:{auto-delete:True, arguments:
{'qpid.alert_count_down':1,'qpid.alert_count_up':3}}}}")
snd = session.sender("threshold-queue")

snd.send("Message1")
snd.send("Message2")
snd.send("Message3")
rcv.fetch()
rcv.fetch()
rcv.fetch()
```

Report a bug

4.10. Deleting Queues

4.10.1. Delete a Queue with qpid-config

The following **qpid-config** command deletes an empty queue:

```
qpid-config del queue queue-name
```

The command will check that the queue is empty before performing the delete, and will report an error and not delete the queue if the queue contains messages.

To delete a queue that contains messages, use the **--force** switch:

```
qpid-config del queue queue-name --force
```

Report a bug

4.10.2. Automatically Deleted Queues

Queues can be configured to *auto-delete*. The broker will delete an auto-delete queue when it has no more subscribers, or if it is auto-delete *and* exclusive, when the declaring session ends.

Applications can delete queues themselves, but if an application fails or loses its connection it may not get the opportunity to clean up its queues. Specifying a queue as auto-delete delegates the responsibility to the broker to clean up the queue when it is no longer needed.

Auto-deleted queues are generally created by an application to *receive* messages, for example: a response queue to specify in the "reply-to" property of a message when requesting information from a service. In this scenario, an application creates a queue for its own use and subscribes it to an exchange. When the consuming application shuts down, the queue is deleted automatically. The queues created by the **qpid-config** utility to receive information from the message broker are an example of this pattern.

A queue configured to **auto-delete** is deleted by the broker after the last consumer has released its subscription to the queue. After the **auto-delete** queue is created, it becomes eligible for deletion as soon as a consumer subscribes to the queue. When the number of consumers subscribed to the queue reaches zero, the queue is deleted.

Here is an example using the Python API to create an auto-delete queue with the name "my-response-queue":

Python

```
responsequeue = session.receiver('my-response-queue; {create:always,
node:{x-declare:{auto-delete:True}}}')
```

Note

Because no bindings are specified in this queue creation, it is bound to the server's **default** exchange: a pre-configured nameless direct exchange.

Custom Timeout

A custom timeout can be configured to provide a grace period before the deletion occurs.

Note

Starting from MRG-M 3.1.0, the C++ client adds a default value of 120 seconds to all durable subscriptions. The qpid python and Java clients do not have a default set, and must be configured manually.

If ***`qpid.auto_delete_timeout:0`*** is specified, the parameter has no effect: setting the parameter to 0 turns off the delayed auto-delete function.

If a timeout of 120 seconds is specified, the broker will wait for 120 seconds after the last consumer disconnects from the queue before deleting it. If a consumer subscribes to the queue within that grace period, the queue is not deleted. This is useful to allow for a consumer to drop its connection and reconnect without losing the information in its queue.

Here is an example using the Python API to create an auto-delete queue with the name "my-response-queue" and an auto-delete timeout of 120 seconds:

Python

```
responsequeue = session.receiver("my-response-queue; {create:always,
node:{x-declare:{auto-delete:True, arguments:
{'qpid.auto_delete_timeout':120}}}}")
```

Be aware that a public auto-deleted queue can be deleted while your application is still sending to it, if your application is not holding it open with a receiver. You will not receive an error because you are sending to an exchange, which continues to exist; however your messages will not go to the now non-existent queue.

If you are publishing to a self-created auto-deleted queue, carefully consider whether using an auto-deleted queue is the correct approach. If the answer is "yes" (it can be useful for tests that clean up after themselves), then subscribe to the queue when you create it. Your subscription will then act as a handle, and the queue will not be deleted until you release it.

Using the Python API:

Python

```
testqueue = session.sender("my-test-queue; {create:always, node:{x-
declare:{auto-delete:True}}}")
testqueuehandle = session.receiver("my-test-queue")
    .....
connection.close()
# testqueuehandle is now released
```

An exception to the requirement that a consumer subscribe and then unsubscribe to invoke the auto-deletion is a queue configured to be **`exclusive`** and **`auto-delete`**; these queues are deleted by the broker when the session that declared the queue ends, since the session that declared the queue is only possible subscriber.

Report a bug

4.10.3. Queue Deletion Checks

When a queue deletion is requested, the following checks occur:

- If ACL is enabled, the broker will check that the user who initiated the deletion has permission to do so.
- If the **`ifEmpty`** flag is passed the broker will raise an exception if the queue is not empty
- If the **`ifUnused`** flag is passed the broker will raise an exception if the queue has subscribers
- If the queue is exclusive the broker will check that the user who initiated the deletion owns the queue

Report a bug

4.11. Producer Flow Control

4.11.1. Flow Control

The broker implements producer flow control on queues that have limits set. This blocks message producers that risk overflowing a destination queue. The queue will become unblocked when enough messages are delivered and acknowledged.

Flow control relies on a reliable link between the sender and the broker. It works by holding off acknowledging sent messages, causing message producers to reach their sender replay buffer capacity and stop sending.

Queues that have been configured with a Limit Policy of type **`ring`** do *not* have queue flow thresholds enabled. These queues deal with reaching capacity through the **`ring`** mechanism. All other queues with limits have two threshold values that are set by the broker when the queue is created:

flow_stop_threshold

the queue resource utilization level that enables flow control when exceeded. Once crossed, the queue is considered in danger of overflow, and the broker will cease acknowledging sent messages to induce producer flow control. Note that *either* queue size or message count capacity utilization can trigger this.

flow_resume_threshold

the queue resource utilization level that disables flow control when dropped below. Once crossed, the queue is no longer considered in danger of overflow, and the broker again acknowledges sent messages. Note that once trigger by either, *both* queue size and message count must fall below this threshold before producer flow control is deactivated.

The values for these two parameters are percentages of the capacity limits. For example, if a queue has a **`qpid.max_size`** of 204800 (200MB), and a **`flow_stop_threshold`** of **`80`**, then the broker will initiate producer flow control if the queue reaches 80% of 204800, or 163840 bytes of enqueued messages.

When the resource utilization of the queue falls below the **`flow_resume_threshold`**, producer flow control is stopped. Setting the **`flow_resume_threshold`** above the **`flow_stop_threshold`** has the obvious consequence of locking producer flow control on, so don't do it.

Report a bug

4.11.2. Queue Flow State

The flow control state of a queue can be determined by the **`flowState`** boolean in the queue's QMF management object. When this is **`true`** flow control is active.

The queue's management object also contains a counter **`flowStoppedCount`** that increments each time flow control becomes active for the queue.

Report a bug

4.11.3. Broker Default Flow Thresholds

The default flow Control Thresholds can be set for the broker using the following two broker options:

- **`--default-flow-stop-threshold`** = flow control activated at this percentage of capacity (size or count)
- **`--default-flow-resume-threshold`** = flow control de-activated at this percentage of capacity (size or count)

For example, the following command starts the broker with flow control set to activate by default at 90% of queue capacity, and deactivate when the queue drops back to 75% capacity:

```
qpidd --default-flow-stop-threshold=90 --default-flow-resume-threshold=75
```

Report a bug

4.11.4. Disable Broker-wide Default Flow Thresholds

To turn off flow control on all queues on the broker by default, start the broker with the default flow control parameters set to 100%:

```
qpidd --default-flow-stop-threshold=100 --default-flow-resume-threshold=100
```

Report a bug

4.11.5. Per-Queue Flow Thresholds

You can set specific flow thresholds for a queue using the following arguments:

`qpid.flow_stop_size`

`integer` flow stop threshold value in bytes.

`qpid.flow_resume_size`

`integer` flow resume threshold value in bytes.

`qpid.flow_stop_count`

`integer` flow stop threshold value as a message count.

`qpid.flow_resume_count`

`integer` flow resume threshold value as a message count.

To disable flow control for a specific queue, set the flow control parameters for that queue to zero.

Report a bug

Chapter 5. Reliably Deliver Messages with Persistence

5.1. Persistent Messages

A persistent message is a message that must not be lost, even if the broker fails.

When a message is marked as persistent *and* sent to a durable queue, it will be written to disk, and resent on restart if the broker fails or shutdowns.

Messages marked as persistent and sent to non-durable queues will not be persisted by the broker.

Note that messages sent using the JMS API are marked persistent by default. If you are sending a message using the JMS API to a durable queue, and do not wish to incur the overhead of persistence, set the message persistence to false.

Messages sent using the C++ API are not persistent by default. To mark a message persistent when using the C++ API, use **`Message.setDurable(true)`** to mark a message as persistent.

Report a bug

5.2. Durable Queues and Guaranteed Delivery

5.2.1. Configure persistence stores

The Red Hat Enterprise Messaging broker enables persistence by default. To verify that persistence is active, make sure that the log shows that the journal is created and the store module initialized when the broker is started. The broker log will contain a line:

```
notice Journal "TplStore": Created
```

Important

If the persistence module is not loaded, messages and the broker state will not be stored to disk, even if the queue is marked durable, and messages are marked persistent.

The **`--store-dir`** command specifies the directory used for the persistence store and any configuration information. The default directory is **`/var/lib/qpidd`** when **`qpidd`** is run as a service, or **`~/.qpidd`** when **`qpidd`** is run from the command line. If **`--store-dir`** is not specified, a subdirectory is created within the directory identified by **`--data-dir`**; if **`--store-dir`** is not specified, and **`--no-data-dir`** is specified, an error is raised.

Important

Only one running broker can access a data directory at a time. If another broker attempts to access the data directory it will fail with an error stating: ***`Exception: Data directory is locked by another process.`***

Report a bug

5.2.2. Durable Queues

By default, the lifetime of a queue is bound to the execution of the server process. When the server shuts down the queues are destroyed, and need to be re-created when the broker is restarted. A *durable queue* is a queue that is automatically re-established after a broker is restarted due to a planned or unplanned shutdown.

When the server shuts down and the queues are destroyed, any messages in those queues are lost. As well as automatic re-creation on server restart, durable queues provide *message persistence* for messages that request it. Messages that are marked as persistent and sent to a durable queue are stored and re-delivered when the durable queue is re-established after a shutdown.

Note that not all messages sent to a durable queue are persistent - only those that are marked as persistent. Note also that marking a message as persistent has no effect if it is sent to a queue that is non-durable. A message must be marked as persistent and sent to a durable queue to be persistent.

Report a bug

5.2.3. Create a durable queue using qpid-config

Use the **`--durable`** option with **`qpid-config add queue`** to create a durable queue. For example:

```
qpid-config add queue --durable durablequeue
```

Report a bug

5.2.4. Mark a message as persistent

A *persistent message* is a message that must not be lost even if the broker fails. To make a message persistent, set the delivery mode to **`PERSISTENT`**. For instance, in C++, the following code makes a message persistent:

```
message.getDeliveryProperties().setDeliveryMode(PERSISTENT);
```

If a persistent message is delivered to a durable queue, it is written to disk when it is placed on the queue.

When a message producer sends a persistent message to an exchange, the broker routes it to any durable queues, and waits for the message to be written to the persistent store, before acknowledging delivery to the message producer. At this point, the durable queue has assumed responsibility for the message, and can ensure that it is not lost even if the broker fails. If a queue is not durable, messages on the queue are not written to disk. If a message is not marked as persistent, it is not written to disk even if it is on a durable queue.

Table 5.1. Persistent Message and Durable Queue Disk States

A persistent message AND durable queue	Written to disk
A persistent message AND non-durable queue	Not written to disk
A non-persistent message AND non-durable queue	Not written to disk
A non-persistent message AND durable queue	Not written to disk

When a message consumer reads a message from a queue, it is not removed from the queue until the consumer acknowledges the message (this is true whether or not the message is persistent or the queue is durable). By acknowledging a message, the consumer takes responsibility for the message, and the queue is no longer responsible for it.

Report a bug

5.2.5. Durable Message State After Restart

When a durable queue is re-established after a restart of the broker, any messages that were marked as persistent and were not reliably delivered before the broker shut down are recovered. The broker does not have information about the delivery status of these messages. They may have been delivered but not acknowledged before the shutdown occurred. To warn receivers that these messages have potentially been previously delivered, the broker sets the **`redelivered`** flag on *all* recovered persistent messages.

Consuming applications should treat the **`redelivered`** flag as a suggestion.

Report a bug

5.3. Message Journal

5.3.1. Journal Description

The term *Messaging Journal*, or *journal*, refers to the on-disk storage for messages.

As its journal implementation, Red Hat Enterprise Messaging 3 uses a *linear store* that dynamically expands as required. There is one journal for each queue; it records each enqueue, dequeue, or transaction event, in order.

The terms *journal* and *store* both refer to the on-disk storage.

In contrast to MRG 2's legacystore module, which has circular disk journals of fixed size, the linear store used by MRG 3 adds files to each queue as needed from a pool of empty files called the Empty File Pool (EFP), and returns files which no longer contain records to the EFP. This allows for arbitrary sized journals, without a size limit.

Best performance is obtained when the EFP is populated with empty files that can be used for journals in advance. However, if the EFP is not present or is empty, the store creates and formats the files as needed (with a performance penalty). The used files are returned to the EFP.

The old store geometry parameters are no longer valid (**`file-size`**, **`num-jfiles`**). The EFP uses a default file size of 2MB per file.

The broker option **`--store-dir`** specifies where the store is located. The broker creates a "**`qls`**" (Qpid linear store) directory under the specified store dir, where it locates the Empty File Pool, the db4 database and the journals. If a specific **`--store-dir`** is not specified, the directory specified by **`--data-dir`** will be used, otherwise the default location is used.

Report a bug

5.3.2. Configuring the Journal

The broker persistence options control a number of journal aspects.

See Also:

- Section 3.3.6, “Persistence Options”

Report a bug

Chapter 6. Increase Message Throughput with Performance Tuning

6.1. Run the JMS Client with real-time Java

To achieve more deterministic behavior, the JMS Client can be run in a Realtime Java environment.

1. The client must be run on a realtime operating system, and supported by your realtime java vendor. Red Hat supports only Sun and IBM implementations.
2. Place the realtime .jar files provided by your vendor in the classpath.
3. Set the following JVM argument:

   ```
   -Dqpid.thread_factory="org.apache.qpid.thread.RealtimeThreadFactory"
   ```

   This ensures that the JMS Client will use **`javax.realtime.RealtimeThread`**s instead of **`java.lang.Thread`**s.

   Optionally, the priority of the Threads can be set using:

   ```
   -Dqpid.rt_thread_priority=30
   ```

   By default, the priority is set at 20.
4. Based on your workload, the JVM will need to be tuned to achieve the best results. Refer to your vendor's JVM tuning guide for more information.

Report a bug

6.2. qpid-latency-test

`qpid-latency-test` is a command-line utility for measuring latency. It is supplied as part of the **`qpid-cpp-client-devel`** package.

Running **`qpid-latency-test`** provides statistics on the performance of your Messaging Server. You can compare the results of **`qpid-latency-test`** with the performance of your application to determine whether your application or the Messaging Server is a performance bottleneck.

`qpid-latency-test --help` provides further information on running the utility.

Report a bug

6.3. Infiniband

6.3.1. Using Infiniband

MRG Messaging connections can use Infiniband, which provides high speed point-to-point bidirectional serial links that can be faster and have much lower latency than TCP connections.

Report a bug

6.3.2. Prerequisites for using Infiniband

The machines running the server and client must each have Infiniband properly installed. In particular:

- The kernel driver and the user space driver for your Infiniband hardware must both be installed.
- Allocate lockable memory for Infiniband.

  By default, the operating system can swap out all user memory. Infiniband requires lockable memory, which can not be swapped out. Each connection requires 8 Megabytes (8192 bytes) of lockable memory.

  To allocate lockable memory, edit **`/etc/security/limits.conf`** to set the limit, which is the maximum amount of lockable memory that a given process can allocate.
- The Infiniband interface must be configured to allow IP over Infiniband. This is used for RDMA connection management.

Report a bug

6.3.3. Configure Infiniband on the Messaging Server

Prerequisites

- The package **`qpid-cpp-server-rdma`** must be installed for Qpid to use RDMA.
- The RDMA plugin, **`rdma.so`**, must be present in the **`plugins`** directory.

Procedure 6.1. Configure Infiniband on the Messaging Server

- **Allocate lockable memory for Infiniband**

  Edit **`/etc/security/limits.conf`** to allocate lockable memory for Infiniband.

  For example, if the user running the server is qpidd, and you wish to support 64 connections (64*8192=524288), add these entries:

```
qpidd soft memlock 524288
qpidd hard memlock 524288
```

Report a bug

6.3.4. Configure Infiniband on a Messaging Client

Prerequisites

- The package **`qpid-cpp-client-rdma`** must be installed.

Procedure 6.2. Configure Infiniband on a Messaging Client

- **Allocate lockable memory for Infiniband**

  Edit **`/etc/security/limits.conf`** to allocate lockable memory.

  To set a limit for all users, for example supporting 16 connections (16*8192=32768), add this entry:

```
* soft memlock 32768
```

If you want to set a limit for a particular user, use the UID for that user when setting the limits:

```
andrew soft memlock 32768
```

Report a bug

Chapter 7. Logging

7.1. Logging in C++

The Qpidd broker and C++ clients can both use environment variables to enable logging. Linux and Windows systems use the same named environment variables and values.

1. Use **QPID_LOG_ENABLE** to set the level of logging you are interested in (***trace***, ***debug***, ***info***, ***notice***, ***warning***, ***error***, or ***critical***):

```
export QPID_LOG_ENABLE="warning+"
```

2. The Qpidd broker and C++ clients use **QPID_LOG_OUTPUT** to determine where logging output should be sent. This is either a file name or the special values ***stderr***, ***stdout***, or ***syslog***:

```
export QPID_LOG_TO_FILE="/tmp/myclient.out"
```

3. From a Windows command prompt, use the following command format to set the environment variables:

```
set QPID_LOG_ENABLE=warning+
set QPID_LOG_TO_FILE=D:\tmp\myclient.out
```

Report a bug

7.2. Change Broker Logging Verbosity

Changes

- New content - added February 2013.

When running qpidd from command line, use **--log-enable** option with the syntax:

```
--log-enable LEVEL[+][:PATTERN]
```

When using a configuration file (**/etc/qpid/qpidd.conf** by default), use the line:

```
log-enable=LEVEL[+][:PATTERN]
```

Notes

- **LEVEL** is one of: **trace debug info notice warning error critical**.
- The "**+**" means log the given severity and any higher severity (without the plus, logging of the given severity only will be enabled).
- **PATTERN** is the scope of the logging change.
- The string in **PATTERN** is matched against the fully-qualified name of the C++ function with the logging statement.

- To see the fully-qualified name of the C++ function with the logging statement, either check the source code or add to the qpid configuration the **`log-function=yes`** option to force qpid broker to log such message.
- So e.g. **`--log-enable debug+:ha`** matches everything in the **`qpid::ha`** module, while e.g. **`--log-enable debug+:broker::Queue::consumeNextMessage`** will enable logging of one particular method only (the **`consumeNextMessage`** method in the given namespace in this example).
- **`PATTERN`** is often set to the module one needs to debug, like **`acl`**, **`amqp_0_10`**, **`broker`**,**`ha`**, **`management`** or **`store`**.
- The option can be used multiple times.
- Be aware that having just one option like "**`log-enable=debug+:ha`**" enables debug logs of ha information, but does not produce any other logs; to add some more verbose logging, add an option like the above and also add the default value: **`log-enable=info+`**

Report a bug

7.3. Change Broker Logging Time Resolution

The time stamp resolution of logging can be changed while the broker is running.

Procedure 7.1. Change Resolution Logging on a Running Broker

1. Edit the file **`/etc/qpid/qpidd.conf`** and add the following:

```
log-time=1
log-enable=info+
log-to-file=/var/lib/qpidd/771830.log
```

2. Launch **`qpid-tool`**.
3. Now that you are running qpid-tool, issue the following command:

```
list broker
```

   You might have to do this a few times before receiving an answer. It can take a while to get all the info from the broker.

4. When you see a response similar to this:

```
114  14:03:39  -          amqp-broker
```

   Use the number (in this example "114") to refer to the broker.

5. Issue the following command, substituting the appropriate number for your broker:

```
call 114 setLogHiresTimestamp 1
```

6. Now look at the log file **`/var/lib/qpidd/771830.log`** and verify that it has started using highres time stamps. You might need to do something to get the broker to log a few more lines, for example start another **`qpid-tool`**.
7. To return the logging to the normal resolution, issue the following command in **`qpid-tool`**:

```
call 114 setLogHiresTimestamp 0
```

8. Now look at the logfile again, and verify that it has stopped using highres.

Report a bug

7.4. Tracking Object Lifecycles

The **[Model]** log category tracks the creation, destruction, and major state changes to Connection, Session, and Subscription objects, and to Exchange, Queue, and Binding objects.

From this set of log messages you can determine which user from which client system address created a connection, what sessions were created on that connection, and what subscriptions were created on those sessions.

Similarly, the exchange-binding-queue objects have enough in their log messages to correlate the interactions between them. The log message for the destruction of an object contains a record of all the management statistics kept for that object. Working through the log records you can attribute broker usage back to specific users.

At **debug** log level are log entries that mirror the corresponding management events. Debug level statements include user names, remote host information, and other references using the user-specified names for the referenced objects.

At **trace** log level are log entries that track the construction and destruction of managed resources. Trace level statements identify the objects using the internal management keys. The trace statement for each deleted object includes the management statistics for that object.

Enabling the Model log

- Use the switch: **--log-enable trace+:Model** to receive both flavors of log.
- Use the switch: **--log-enable debug+:Model** for a less verbose log.

Managed Objects in the logs

All managed objects are included in the trace log. The debug log has information for: **Connection, Queue, Exchange, Binding, Subscription**.

The following are actual log file data sorted and paired with the corresponding management Event captured with **qpid-printevents**.

1. Connection

Create connection

```
event: Fri Jul 13 17:46:23 2012 org.apache.qpid.broker:clientConnect rhost=
[::1]:5672-[::1]:34383 user=anonymous
debug: 2012-07-13 13:46:23 [Model] debug Create connection. user:anonymous
rhost:[::1]:5672-[::1]:34383
trace: 2012-07-13 13:46:23 [Model] trace Mgmt create connection. id:
[::1]:5672-[::1]:34383
```

Delete connection

```
event: Fri Jul 13 17:46:23 2012 org.apache.qpid.broker:clientDisconnect
rhost=[::1]:5672-[::1]:34383 user=anonymous
debug: 2012-07-13 13:46:23 [Model] debug Delete connection. user:anonymous
rhost:[::1]:5672-[::1]:34383
trace: 2012-07-13 13:46:29 [Model] trace Mgmt delete connection. id:
[::1]:5672-[::1]:34383
Statistics: {bytesFromClient:1451, bytesToClient:892, closing:False,
framesFromClient:25, framesToClient:21, msgsFromClient:1, msgsToClient:1}
```

2. Session

Create session

```
event: TBD
debug: TBD
trace: 2012-07-13 13:46:09 [Model] trace Mgmt create session. id:18f52c22-
efc5-4c2f-bd09-902d2a02b948:0
```

Delete session

```
event: TBD
debug: TBD
trace: 2012-07-13 13:47:13 [Model] trace Mgmt delete session. id:18f52c22-
efc5-4c2f-bd09-902d2a02b948:0
Statistics: {TxnCommits:0, TxnCount:0, TxnRejects:0, TxnStarts:0,
clientCredit:0, unackedMessages:0}
```

3. Exchange

Create exchange

```
event: Fri Jul 13 17:46:34 2012 org.apache.qpid.broker:exchangeDeclare
disp=created exName=myE exType=topic durable=False args={} autoDel=False
rhost=[::1]:5672-[::1]:34384 altEx= user=anonymous
debug: 2012-07-13 13:46:34 [Model] debug Create exchange. name:myE
user:anonymous rhost:[::1]:5672-[::1]:34384 type:topic alternateExchange:
durable:F
trace: 2012-07-13 13:46:34 [Model] trace Mgmt create exchange. id:myE
```

Delete exchange

```
event: Fri Jul 13 18:19:33 2012 org.apache.qpid.broker:exchangeDelete
exName=myE rhost=[::1]:5672-[::1]:37199 user=anonymous
debug: 2012-07-13 14:19:33 [Model] debug Delete exchange. name:myE
user:anonymous rhost:[::1]:5672-[::1]:37199
trace: 2012-07-13 14:19:42 [Model] trace Mgmt delete exchange. id:myE
Statistics: {bindingCount:0, bindingCountHigh:0, bindingCountLow:0,
byteDrops:0, byteReceives:0, byteRoutes:0, msgDrops:0, msgReceives:0,
msgRoutes:0, producerCount:0, producerCountHigh:0, producerCountLow:0}
```

4. Queue

Create queue

```
event: Fri Jul 13 18:19:35 2012 org.apache.qpid.broker:queueDeclare
disp=created durable=False args={} qName=myQ autoDel=False rhost=[::1]:5672-
[::1]:37200 altEx= excl=False user=anonymous
debug: 2012-07-13 14:19:35 [Model] debug Create queue. name:myQ
user:anonymous rhost:[::1]:5672-[::1]:37200 durable:F owner:0 autodelete:F
alternateExchange:
trace: 2012-07-13 14:19:35 [Model] trace Mgmt create queue. id:myQ
```

Delete queue

```
event: Fri Jul 13 18:19:37 2012 org.apache.qpid.broker:queueDelete
user=anonymous qName=myQ rhost=[::1]:5672-[::1]:37201
debug: 2012-07-13 14:19:37 [Model] debug Delete queue. name:myQ
user:anonymous rhost:[::1]:5672-[::1]:37201
trace: 2012-07-13 14:19:42 [Model] trace Mgmt delete queue. id:myQ
Statistics: {acquires:0, bindingCount:0, bindingCountHigh:0,
bindingCountLow:0, byteDepth:0, byteFtdDepth:0, byteFtdDequeues:0,
byteFtdEnqueues:0, bytePersistDequeues:0, bytePersistEnqueues:0,
byteTotalDequeues:0, byteTotalEnqueues:0, byteTxnDequeues:0,
byteTxnEnqueues:0, consumerCount:0, consumerCountHigh:0, consumerCountLow:0,
discardsLvq:0, discardsOverflow:0, discardsPurge:0, discardsRing:0,
discardsSubscriber:0, discardsTtl:0, flowStopped:False, flowStoppedCount:0,
messageLatencyAvg:0, messageLatencyCount:0, messageLatencyMax:0,
messageLatencyMin:0, msgDepth:0, msgFtdDepth:0, msgFtdDequeues:0,
msgFtdEnqueues:0, msgPersistDequeues:0, msgPersistEnqueues:0,
msgTotalDequeues:0, msgTotalEnqueues:0, msgTxnDequeues:0, msgTxnEnqueues:0,
releases:0, reroutes:0, unackedMessages:0, unackedMessagesHigh:0,
unackedMessagesLow:0}
```

5. Binding

Create binding

```
event: Fri Jul 13 17:46:45 2012 org.apache.qpid.broker:bind exName=myE args=
{} qName=myQ user=anonymous key=myKey rhost=[::1]:5672-[::1]:34385
debug: 2012-07-13 13:46:45 [Model] debug Create binding. exchange:myE
queue:myQ key:myKey user:anonymous rhost:[::1]:5672-[::1]:34385
trace: 2012-07-13 13:46:23 [Model] trace Mgmt create binding.
id:org.apache.qpid.broker:exchange:,org.apache.qpid.broker:queue:myQ,myQ
```

Delete binding

```
event: Fri Jul 13 17:47:06 2012 org.apache.qpid.broker:unbind user=anonymous
exName=myE qName=myQ key=myKey rhost=[::1]:5672-[::1]:34386
debug: 2012-07-13 13:47:06 [Model] debug Delete binding. exchange:myE
queue:myQ key:myKey user:anonymous rhost:[::1]:5672-[::1]:34386
trace: 2012-07-13 13:47:09 [Model] trace Mgmt delete binding.
id:org.apache.qpid.broker:exchange:myE,org.apache.qpid.broker:queue:myQ,myKe
y
Statistics: {msgMatched:0}
```

6. Subscription

Create subscription

```
event: Fri Jul 13 18:19:28 2012 org.apache.qpid.broker:subscribe dest=0
args={} qName=b78b1818-7a20-4341-a253-76216b40ab4a:0.0 user=anonymous
excl=False rhost=[::1]:5672-[::1]:37198
debug: 2012-07-13 14:19:28 [Model] debug Create subscription.
queue:b78b1818-7a20-4341-a253-76216b40ab4a:0.0 destination:0 user:anonymous
rhost:[::1]:5672-[::1]:37198 exclusive:F
trace: 2012-07-13 14:19:28 [Model] trace Mgmt create subscription.
id:org.apache.qpid.broker:session:b78b1818-7a20-4341-a253-
76216b40ab4a:0,org.apache.qpid.broker:queue:b78b1818-7a20-4341-a253-
76216b40ab4a:0.0,0
```

Delete subscription

```
event: Fri Jul 13 18:19:28 2012 org.apache.qpid.broker:unsubscribe dest=0
rhost=[::1]:5672-[::1]:37198 user=anonymous
debug: 2012-07-13 14:19:28 [Model] debug Delete subscription. destination:0
user:anonymous rhost:[::1]:5672-[::1]:37198
trace: 2012-07-13 14:19:32 [Model] trace Mgmt delete subscription.
id:org.apache.qpid.broker:session:b78b1818-7a20-4341-a253-
76216b40ab4a:0,org.apache.qpid.broker:queue:b78b1818-7a20-4341-a253-
76216b40ab4a:0.0,0
Statistics: {delivered:1}
```

Report a bug

Chapter 8. Secure Your Connections and Resources

8.1. Simple Authentication and Security Layer - SASL

8.1.1. SASL - Simple Authentication and Security Layer

MRG Messaging uses Simple Authentication and Security Layer (SASL) for identifying and authorizing incoming connections to the broker, as mandated in the AMQP specification. SASL provides a variety of authentication methods. The Messaging Broker can be configured to allow any combination of the available SASL mechanisms. Clients can negotiate with the Messaging Broker to find a SASL mechanism that both can use.

MRG Messaging clients (with the exception of the JMS client) and the broker use the Cyrus SASL library to allow for a full SASL implementation.

Report a bug

8.1.2. SASL Support in Windows Clients

The Windows Qpid C++ and C# clients support only **`ANONYMOUS`** and **`PLAIN`** and **`EXTERNAL`** authentication mechanisms.

No other SASL mechanisms are supported by Windows at this time.

If no sasl-mechanism is specified, the default chosen mechanism will usually differ between Windows and Linux.

Report a bug

8.1.3. SASL Mechanisms

Changes

- Updated April 2013.

The SASL authentication mechanisms allowed by the broker are controlled by the file **`/etc/sasl2/qpidd.conf`** on the broker. To narrow the allowed mechanisms to a smaller subset, edit this file and remove mechanisms.

Important

The PLAIN authentication mechanism sends passwords in cleartext. If using this mechanism, for complete security using Security Services Library (SSL) is recommended.

SASL Mechanisms

ANONYMOUS

Clients are able to connect anonymously.

Note that when the broker is started with **`auth=no`**, authentication is disabled. **`PLAIN`** and **`ANONYMOUS`** authentication mechanisms are available as *identification mechanisms*, but they have no authentication value.

PLAIN

Passwords are passed in plain text between the client and the broker. This is not a secure mechanism, and should be used in development environments only. If PLAIN is used in production, it should only be used over SSL connections, where the SSL encryption of the transport protects the password.

Note that when the broker is started with **`auth=no`**, authentication is disabled. The **`PLAIN`** and **`ANONYMOUS`** authentication mechanisms are available as *identification mechanisms*, but they have no authentication value.

DIGEST-MD5

MD5 hashed password exchange using HTTP headers. This is a medium strength security protocol.

CRAM-MD5

A challenge-response protocol using MD5 encryption.

KERBEROS/GSSAPI

The Generic Security Service Application Program Interface (GSSAPI) is a framework that allows for the connection of different security providers. By far the most frequently used is Kerberos. GSSAPI security provides centralized management of security, including single sign-on, opaque token exchange, and transport security.

EXTERNAL

EXTERNAL SASL authentication uses an SSL-encrypted connection between the client and the server. The client presents a certificate to encrypt the connection, and this certificate contains both the cryptographic key for the connection and the identity of the client.

Report a bug

8.1.4. SASL Mechanisms and Packages

The following table lists the **`cyrus-sasl-*`** package(s) that need to be installed on the server for each authentication mechanism.

Table 8.1.

Method	Package	`/etc/sasl2/qpidd.conf` entry
ANONYMOUS	-	-
PLAIN	**`cyrus-sasl-plain`**	**`mech_list: PLAIN`**
DIGEST-MD5	**`cyrus-sasl-md5`**	**`mech_list: DIGEST-MD5`**
CRAM-MD5	**`cyrus-sasl-md5`**	**`mech_list: CRAM-MD5`**
KERBEROS/GSSAPI	**`cyrus-sasl-gssapi`**	**`mech_list: GSSAPI`**
EXTERNAL	-	**`mech_list: EXTERNAL`**

Report a bug

8.1.5. Configure SASL using a Local Password File

The local SASL database is used by the PLAIN, DIGEST-MD5, or CRAM-MD5 SASL authentication mechanisms. The GSSAPI/KERBEROS mechanism authenticates clients against the Kerberos user database, and the EXTERNAL mechanism uses the client identity in the SSL certificate used by the client to encrypt the connection.

Procedure 8.1. Configure SASL using a Local Password File

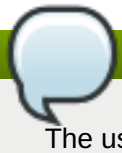
1. Add new users to the database by using the **saslpasswd2** command. The User ID for authentication and ACL authorization uses the form ***user-id*@*domain***.

   Ensure that the correct realm has been set for the broker. This can be done by editing the configuration file or using the **-u** option. The default realm for the broker is ***QPID***.

   ```
   # saslpasswd2 -f /var/lib/qpidd/qpidd.sasldb -u QPID new_user_name
   ```

2. Existing user accounts can be listed by using the **-f** option:

   ```
   # sasldblistusers2 -f /var/lib/qpidd/qpidd.sasldb
   ```

   Note

   The user database at **/var/lib/qpidd/qpidd.sasldb** is readable only by the **qpidd** user. If you start the broker from a user other than the **qpidd** user, you will need to either modify the configuration file, or turn authentication off.

   Note also that this file must be readable by the **qpidd** user. If you delete and recreate this file, make sure the qpidd user has read permissions, or authentication attempts will fail.

3. To switch authentication on or off, use the **auth yes|no** option when you start the broker:

   ```
   # /usr/sbin/qpidd --auth yes

   # /usr/sbin/qpidd --auth no
   ```

   You can also set authentication to be on or off by adding the appropriate line to to the **/etc/qpid/qpidd.conf** configuration file:

   ```
   auth=no

   auth=yes
   ```

   The SASL configuration file is in **/etc/sasl2/qpidd.conf** for Red Hat Enterprise Linux.

Report a bug

8.1.6. Configure SASL with ACL

1. To start using the ACL, specify the path and filename using the **--acl-file** command. The filename should have a **.acl** extension:

   ```
   $ qpidd --acl-file ./aclfilename.acl
   ```

2. Optionally, you can limit the number of active connections per user with the **`--connection-limit-per-user`** and **`--connection-limit-per-ip`** commands. These limits can only be enforced if the **`--acl-file`** command is specified.

3. You can now view the file with the **`cat`** command and edit it in your preferred text editor. If the path and filename is not found, **`qpidd`** will fail to start.

Report a bug

8.1.7. Configure Kerberos 5

Kerberos uses the GSSAPI (Generic Security Services Application Program Interface) authentication mechanism on the broker to authenticate with a Kerberos server.

Both the MRG Messaging broker and users are principals of the Kerberos server, which means that they are both clients of the Kerberos authentication services.

Note

The following instructions make use of example domain names and Kerberos realms. To follow these instructions you must have a Kerberos server configured and use the appropriate domain names and Kerberos realm for your network environment.

To use Kerberos, both the broker and each user must be authenticated on the Kerberos server:

1. Install the Kerberos workstation software and Cyrus SASL GSSAPI on each machine that runs a qpidd broker or a qpidd messaging client:

```
$ sudo yum install cyrus-sasl-gssapi krb5-workstation
```

2. Change the mech_list line in **`/etc/sasl2/qpidd.conf`** to:

```
mech_list: GSSAPI
```

3. Add the following lines to **`/etc/qpid/qpidd.conf`**:

```
auth=yes
realm=QPID
```

4. Register the Qpid broker in the Kerberos database.

   Traditionally, a Kerberos principal is divided into three parts: the primary, the instance, and the realm. A typical Kerberos V5 has the format **`primary/instance@REALM`**. For a broker, the primary is **`qpidd`**, the instance is the fully qualified domain name, and the REALM is the Kerberos domain realm. By default, this realm is **`QPID`**, but a different realm can be specified in qpid.conf per the following example.

```
realm=EXAMPLE.COM
```

For instance, if the fully qualified domain name is **dublduck.example.com** and the Kerberos domain realm is **EXAMPLE.COM**, then the principal name is **qpidd/dublduck.example.com@EXAMPLE.COM**.

```
FDQN=`hostname --fqdn`
REALM="EXAMPLE.COM"
kadmin -r $REALM  -q "addprinc -randkey -clearpolicy qpidd/$FQDN"
```

Now create a Kerberos keytab file for the broker. The broker must have read access to the keytab file. The following script creates a keytab file and allows the broker read access:

```
QPIDD_GROUP="qpidd"
kadmin -r $REALM  -q "ktadd -k /etc/qpidd.keytab qpidd/$FQDN@$REALM"
chmod g+r /etc/qpidd.keytab
chgrp $QPIDD_GROUP /etc/qpidd.keytab
```

The default location for the keytab file is **/etc/krb5.keytab**. If a different keytab file is used, the KRB5_KTNAME environment variable must contain the name of the file as the following example shows.

```
export KRB5_KTNAME=/etc/qpidd.keytab
```

If this is correctly configured, you can now enable Kerberos support on the broker by setting the **auth** and **realm** options in **/etc/qpid/qpidd.conf**:

```
CDATA[# /etc/qpid/qpidd.conf
auth=yes
realm=EXAMPLE.COM
```

Restart the broker to activate these settings.

5. Make sure that each Qpid user is registered in the Kerberos database, and that Kerberos is correctly configured on the client machine. The Qpid user is the account from which a Qpid messaging client is run. If it is correctly configured, the following command should succeed:

```
$ kinit user@REALM.COM
```

6. **Additional configuration for Java JMS clients**

   Java JMS clients require a few additional steps.

   a. The Java JVM must be run with the following arguments:

      -Djavax.security.auth.useSubjectCredsOnly=false

      Forces the SASL GSSAPI client to obtain the Kerberos credentials explicitly instead of obtaining from the "subject" that owns the current thread.

      -Djava.security.auth.login.config=myjas.conf

      Specifies the jass configuration file. Here is a sample JASS configuration file:

```
com.sun.security.jgss.initiate {
    com.sun.security.auth.module.Krb5LoginModule required
useTicketCache=true;
};
```

-Dsun.security.krb5.debug=true

Enables detailed debug info for troubleshooting

b. The client Connection URL must specify the following Kerberos-specific broker properties:

- **`sasl_mechs`** must be set to **`GSSAPI`**.
- **`sasl_protocol`** must be set to the principal for the qpidd broker, e.g. **`qpidd`**/
- **`sasl_server`** must be set to the host for the SASL server, e.g. **`sasl.com`**.

Here is a sample connection URL for a Kerberos connection:

```
amqp://guest@clientid/testpath?brokerlist='tcp://localhost:5672?
sasl_mechs='GSSAPI'&sasl_protocol='qpidd'&sasl_server='<server-
host-name>''
```

Report a bug

8.2. Configuring TLS/SSL

8.2.1. Encryption Using SSL

Encryption and certificate management for **`qpidd`** is provided by Mozilla's Network Security Services Library (NSS).

See Also:

- Appendix B, *OpenSSL Certificate Reference*

Report a bug

8.2.2. A Note on Installing Client Certificates

Versions of Red Hat Enterprise Linux prior to 6.4 require a restart of the broker to load new client certificates that are added to the certificate database.

Red Hat Enterprise Linux 6.5 contains a modification that allows client certificates to be added to the database and read by the broker without restarting. This modification is available for Red Hat Enterprise Linux 6.4 as a hot fix.

See Also:

- Appendix B, *OpenSSL Certificate Reference*

Report a bug

8.2.3. Enable SSL on the Broker

Changes

- Updated April 2013.

1. You will need a certificate that has been signed by a Certification Authority (CA). This certificate will also need to be trusted by your client. If you require client authentication in addition to server authentication, the clients certificate will also need to be signed by a CA and trusted by the broker.

   The certificate database is created and managed by the Mozilla Network Security Services (NSS) **`certutil`** tool. Information on this utility can be found on the Mozilla website, including tutorials on setting up and testing SSL connections. The certificate database will generally be password protected. The safest way to specify the password is to place it in a protected file, use the password file when creating the database, and specify the password file with the **`ssl-cert-password-file`** option when starting the broker.

   The following script shows how to create a certificate database using certutil:

   ```
   mkdir ${CERT_DIR}
   certutil -N -d ${CERT_DIR} -f ${CERT_PW_FILE}
   certutil -S -d ${CERT_DIR} -n ${NICKNAME} -s "CN=${NICKNAME}" -t
   "CT,," -x -f ${CERT_PW_FILE} -z /usr/bin/certutil
   ```

   When starting the broker, set **`ssl-cert-password-file`** to the value of *${CERT_PW_FILE}*, set **`ssl-cert-db`** to the value of *${CERT_DIR}*, and set **`ssl-cert-name`** to the value of *${NICKNAME}*.

2. The following SSL options can be used when starting the broker:

   `--ssl-use-export-policy`

   Use NSS export policy. When this option is specified, the server will conform with US export restrictions on encryption using the NSS export policy. When it is not specified, the server will use the domestic policy. Refer to the Mozilla SSL Export Policy Functions documentation for more details.

   `--ssl-cert-password-file` ***`PATH`***

   Required. Plain-text file containing password to use for accessing certificate database.

   `--ssl-cert-db` ***`PATH`***

   Required. Path to directory containing certificate database.

   `--ssl-cert-name` ***`NAME`***

   Name of the certificate to use. Default is **`localhost.localdomain`**.

   `--ssl-port` ***`NUMBER`***

   Port on which to listen for SSL connections. If no port is specified, port 5671 is used.

   If the SSL port chosen is the same as the port for non-SSL connections (i.e. if the **`--ssl-port`** and **`--port`** options are the same), both SSL encrypted and unencrypted connections can be established to the same port. However in this configuration there is no support for IPv6.

   `--ssl-require-client-authentication`

   Require SSL client authentication (i.e. verification of a client certificate) during the SSL handshake. This occurs before SASL authentication, and is independent of SASL.

This option enables the **EXTERNAL** SASL mechanism for SSL connections. If the client chooses the **EXTERNAL** mechanism, the client's identity is taken from the validated SSL certificate, using the **CN**, and appending any **DC**'s to create the domain. For instance, if the certificate contains the properties **CN=bob**, **DC=acme**, **DC=com**, the client's identity is **bob@acme.com**.

If the client chooses a different SASL mechanism, the identity take from the client certificate will be replaced by that negotiated during the SASL handshake.

--ssl-sasl-no-dict

Do not accept SASL mechanisms that can be compromised by dictionary attacks. This prevents a weaker mechanism being selected instead of **EXTERNAL**, which is not vulnerable to dictionary attacks.

--require-encryption

This will cause **qpidd** to only accept encrypted connections. This means only clients with EXTERNAL SASL on the SSL-port, or with GSSAPI on the TCP port.

Report a bug

8.2.4. Export an SSL Certificate for Clients

When SSL is enabled on a server, the clients require a copy of the SSL certificate to establish a secure connection.

The following example commands can be used to export a client certificate and the private key from the broker's NSS database:

```
pk12util -o <p12exportfile> -n <certname> -d <certdir> -w <p12filepwfile>

openssl pkcs12 -in <p12exportfile> -out <clcertname> -nodes -clcerts -passin
pass:<p12pw>
```

For more information on SSL commands and options, refer to the OpenSSL Documentation. On Red Hat Enterprise Linux type: **man openssl**.

Report a bug

8.2.5. Enable SSL on Windows

The following set of procedures export the SSL certificate from the broker and install it on Windows machines to enable SSL connections between clients running on Windows and the broker.

Procedure 8.2. Create SSL certificates on the broker

1. Execute the following commands on the broker to export a certificate:

```
# cd /var/lib/qpidd

# mkdir qpid_nss_db
# cd qpid_nss_db
# ls
# echo password > ssl_pw_file
# cat ssl_pw_file
```

```
password

# certutil -S -d . -n qrootCA -s "CN=qrootCA" -t "CT,," -x -m 1000 -v
120 -f ssl_pw_file
# certutil -S -n "fully-qualified-server-name.com" -s "CN="fully-
qualified-server-name.com -c qrootCA -t ",," -m 1001 -v 120 -d . -f
ssl_pw_file
# certutil -S -n client -s "CN=client" -t ",," -m 1005 -v 120 -c
qrootCA -d . -f ssl_pw_file
# pk12util -d . -o client.p12 -n client
Enter Password or Pin for "NSS Certificate DB":
Enter Password or Pin for "NSS Certificate DB":
Enter password for PKCS12 file:
Re-enter password:
pk12util: PKCS12 EXPORT SUCCESSFUL
# openssl pkcs12 -in client.p12 -out client.pem -nodes -clcerts
Enter Import Password:
MAC verified OK
```

2. Verify that the files exist:

```
# ls
cert8.db  client.p12  client.pem  key3.db  secmod.db  ssl_pw_file
```

Procedure 8.3. Copy the `qpid_nss_db` folder to other broker machines and set `qpidd` as its owner

1. Execute the following commands on the other brokers to copy the files from the first broker:

```
# scp -r qpid_nss_db root@other-broker.com:/var/lib/qpidd
# chown -R qpidd:qpidd qpid_nss_db
```

2. Verify the files and their permissions:

```
# ll
total 89896
-rw-r-----. 1 qpidd qpidd        0 Jul 16 06:27 lock
-rw-r--r--. 1 qpidd qpidd 91989014 Nov  1 06:52 qpidd.log
-rw-------. 1 qpidd qpidd    12288 Oct  7 05:32 qpidd.sasldb
drwxr-xr-x. 2 qpidd qpidd     4096 Nov  6 04:32 qpid_nss_db
-rw-r-----. 1 qpidd qpidd       37 Jul 16 06:27 systemId
```

Procedure 8.4. Modify broker configuration file

- Edit the broker configuration file **`/etc/qpid/qpidd.conf`**:

```
ssl-require-client-authentication=no
log-to-file=/var/lib/qpidd/qpidd.log
ssl-port=5671
log-enable=info+
ssl-cert-password-file=/var/lib/qpidd/qpid_nss_db/ssl_pw_file
ssl-cert-name=fully-qualified-server-name.com
auth=no
ssl-cert-db=/var/lib/qpidd/qpid_nss_db
```

Procedure 8.5. Start the broker

- Start the broker and verify that it is listening on the SSL port:

```
# service qpidd restart
Stopping Qpid AMQP daemon:              [  OK  ]
Starting Qpid AMQP daemon:              [  OK  ]

# netstat -nap | grep qpidd
tcp        0      0 0.0.0.0:5671                0.0.0.0:*
LISTEN      25184/qpidd
tcp        0      0 0.0.0.0:5672                0.0.0.0:*
LISTEN      25184/qpidd
tcp        0      0 :::5671                     :::*
LISTEN      25184/qpidd
tcp        0      0 :::5672                     :::*
LISTEN      25184/qpidd
```

Procedure 8.6. Create a folder to export onto Windows machines

1. Execute the following instructions to:
   - Create a folder to export onto Windows machines
   - Create a new password file in .txt format
   - Export certification authority certificate to .cer format
   - Export client certificate to .pfx format

```
# mkdir windir
# echo password2 > windir/win_pw_file.txt
# cat windir/win_pw_file.txt
password2
# certutil -L -d qpid_nss_db -n qrootCA -f ssl_pw_file -a >
windir/qrootCA.cer
# pk12util -d qpid_nss_db -n client  -k qpid_nss_db/ssl_pw_file -w
windir/win_pw_file.txt -o windir/client.pfx
        pk12util: PKCS12 EXPORT SUCCESSFUL
```

2. Verify that the files exist:

```
# ls windir
client.pfx  qrootCA.cer  win_pw_file.txt
```

Procedure 8.7. Copy files to Windows machine

- Copy the **`windir`** folder onto the Windows machine.

Optional Pathway: GUI or Command-line

The following procedure, to install the Certificate on the Windows machine has two options - using the GUI, or using the command-line.

Procedure 8.8. Install Certification Authority - GUI

1. On the Windows machine, run **mmc**
2. Click **File** / **Add/Remove Snap-in...**
3. Select **Certificates** -> **Add** -> **Computer account** -> **Local computer** -> **Finish** -> **OK**
4. In the console unpack Certificates (Local Computer)
5. Right click on Trusted Root Certification Authorities, and select **All Tasks/Import...**
6. Set the path to the **qrootCA.cer** file, select Trusted Root Certification Authorities certificate store, confirm the action and save the console settings.

Procedure 8.9. Install Certification Authority - Command-line

- Execute the following command to import the certificate at the command-line:

```
certmgr.exe -add -c C:\windir\qrootca.cer -s -r localMachine root
```

Procedure 8.10. Test connection

- Execute the following at the command line to test the connection (no environment variables must be set):

```
C:\qpid_VS2008\bin\Release>spout.exe --broker broker-server.com:5671 --
connection-options {transport:ssl} "amq.topic"
```

Optional Pathway - Certificate Installed or Specified via Environment

You can install the certificate in the Windows machine certificate store, or specify it via environment variables.

Procedure 8.11. Install Certificate in Windows Certificate Store

Follow these instructions to install the client certificate **client.pfx** into Current User/Personal certificate store:

1. Run **mmc**
2. Click **File** / **Add/Remove Snap-in...**
3. Select **Certificates** -> **Add>** -> **My user account** -> **Finish** -> **OK**
4. In the console unpack **Certificates - Current User**
5. Right click on **Personal**.
6. Select **All Tasks / Import**.
7. Assign path to the **client.pfx** file
8. Click on **Next**.
9. Type a password from **win_pw_file.txt** (*password2* in our case).
10. Choose **Certificate Store Personal** and save the console settings.
11. Modify broker configuration to require client authentication and restart it .
12. Set up environment variables:

```
>set QPID_SSL_CERT_STORE=My
>set QPID_SSL_CERT_NAME=client
```

13. Test it by sending a message:

```
>C:\qpid_VS2008\bin\Release>spout.exe --broker broker-server.com:5671
--connection-options {transport:ssl,sasl-mechanisms:EXTERNAL}
amq.topic
```

Procedure 8.12. Specify Certificate via Environment

1. Set up environmental variables on the Windows machine:

```
>set QPID_SSL_CERT_FILENAME=<path_to_the_client.pfx>
>set QPID_SSL_CERT_PASSWORD_FILE=<path_to_the_win_pw_file.txt>
>set QPID_SSL_CERT_NAME=client
```

   For example:

```
>C:\qpid_VS2008\bin\Release>set
QPID_SSL_CERT_FILENAME=C:\windir\client.pfx

>C:\qpid_VS2008\bin\Release>set
QPID_SSL_CERT_PASSWORD_FILE=C:\windir\win_pw_fil
e.txt

>C:\qpid_VS2008\bin\Release>set QPID_SSL_CERT_NAME=client
```

2. Test it by sending a message:

```
C:\qpid_VS2008\bin\Release>spout.exe --broker broker-server.com:5671 -
-connection-options {transport:ssl,sasl-mechanisms:EXTERNAL} amq.topic
```

Report a bug

8.2.6. Enable SSL in C++ Clients

The following options can be specified for C++ clients using environment variables:

Table 8.2. SSL Client Environment Variables for C++ clients

SSL Client Options for C++ clients	
QPID_SSL_USE_EXPORT_POLICY	Use NSS export policy
QPID_SSL_CERT_PASSWORD_FILE *PATH*	File containing password to use for accessing certificate database
QPID_SSL_CERT_DB *PATH*	Path to directory containing certificate database
QPID_SSL_CERT_NAME *NAME*	Name of the certificate to use. When SSL client authentication is enabled, a certificate name should normally be provided.

When using SSL connections, clients must specify the location of the certificate database, a directory that contains the client's certificate and the public key of the Certificate Authority. This can be done by setting the

environment variable **QPID_SSL_CERT_DB** to the full pathname of the directory. If a connection uses SSL client authentication, the client's password is also needed - the password should be placed in a protected file, and the **QPID_SSL_CERT_PASSWORD_FILE** variable should be set to the location of the file containing this password.

To open an SSL enabled connection in the Qpid Messaging API, set the ***transport*** connection option to ***ssl***.

See Also:

- Section 8.2.4, “Export an SSL Certificate for Clients”
- Appendix B, *OpenSSL Certificate Reference*

Report a bug

8.2.7. Enable SSL in Java Clients

1. For both server and client authentication, import the trusted CA to your trust store and keystore and generate keys for them. Create a certificate request using the generated keys and then create a certificate using the request. You can then import the signed certificate into your keystore. Pass the following arguments to the Java JVM when starting your client:

```
-Djavax.net.ssl.keyStore=/home/bob/ssl_test/keystore.jks
-Djavax.net.ssl.keyStorePassword=password
-Djavax.net.ssl.trustStore=/home/bob/ssl_test/certstore.jks
-Djavax.net.ssl.trustStorePassword=password
```

2. For server side authentication only, import the trusted CA to your trust store and pass the following arguments to the Java JVM when starting your client:

```
-Djavax.net.ssl.trustStore=/home/bob/ssl_test/certstore.jks
-Djavax.net.ssl.trustStorePassword=password
```

3. Java clients must use the SSL option in the connection URL to enable SSL encryption, per the following example.

```
amqp://username:password@clientid/test?
brokerlist='tcp://localhost:5672?ssl='true''
```

4. If you need to debug problems in an SSL connection, enable Java's SSL debugging by passing the argument **-Djavax.net.debug=ssl** to the Java JVM when starting your client.

See Also:

- Section 8.2.4, “Export an SSL Certificate for Clients”
- Appendix B, *OpenSSL Certificate Reference*

Report a bug

8.2.8. Enable SSL in Python Clients

To use SSL with the Python client *either*:

1. Use a URL of the form **amqps://<*host*>:<*port*>**, where *host* is the brokers hostname and *port* is the SSL port (usually 5671), *or*
2. Set the '**transport**' attribute of the connection to "**ssl**".

The Python client has some limitations in SSL functionality:

Server authentication must be demanded, and the client name must be explicitly provided when using the **EXTERNAL** SASL mechanism for authentication.

- The Python clients has an optional parameter **ssl_trustfile** (see Python SSL Parameters). When this parameter is specified, trust store validation of the certificate is performed.
- The Python client matches the server's SSL certificate against the connection hostname when the optional parameter **ssl_trustfile** is supplied.
- When using the EXTERNAL SASL mechanism for authentication, you must provide the client name in the connection string. This client name provided in the connection string must match the identity of the SSL certificate. Missing either these two will cause the connection to fail: by not providing the client name in the connection string, or providing a client name that does match the identity of the SSL certificate.

Python SSL Parameters

The QPID Python client accepts the following SSL-related configuration parameters:

- **ssl_certfile** - the path to a file that contains the PEM-formatted certificate used to identify the local side of the connection (the client). This is needed if the server requires client-side authentication.
- **ssl_keyfile** - In some cases the client's private key is stored in the same file as the certificate (i.e. ssl_certfile). If the **ssl_certfile** does not contain the client's private key, this parameter must be set to the path to a file containing the private key in PEM file format.
- **ssl_skip_hostname_check** - When set to true the connection hostname verification against the server certificate is skipped.
- **ssl_trustfile** - this parameter contains a path to a PEM-formatted file containing a chain of trusted Certificate Authority (CA) certificates. These certificates are used to authenticate the remote server.
- These parameters are passed as arguments to the **qpid.Connection()** object when it is constructed. For example:

```
Connection("amqps://client@127.0.0.1:5671",
ssl_certfile="/path/to/certfile", ssl_keyfile="/path/to/keyfile")
```

See Also:

- Section 8.2.4, “Export an SSL Certificate for Clients”
- Appendix B, *OpenSSL Certificate Reference*

Report a bug

8.3. Authorization

8.3.1. Access Control List (ACL)

In MRG Messaging, Authorization specifies which actions can be performed by each authenticated user using an Access Control List (ACL).

Report a bug

8.3.2. Default ACL File

In versions up to 2.2, the location of the default ACL file is **/etc/qpidd.acl**.

From version 2.3, the default ACL file is relocated to **/etc/qpid/qpidd.acl**. Unmodified existing installations will continue to use the previous ACL file and location, while any new installations will use the new default location and file.

Report a bug

8.3.3. Load an Access Control List (ACL)

Use the **--acl-file** command to load the access control list. The filename should have a **.acl** extension:

```
$ qpidd --acl-file ./aclfilename.acl
```

Report a bug

8.3.4. Reloading the ACL

You can reload the ACL without restarting the broker using a QMF method, either using **qpid-tool** or from program code.

Reload the ACL using qpid-tool

You need to use **qpid-tool** with a account with sufficient privileges to reload the ACL.

1. Start **qpid-tool**:

```
$ qpid-tool admin/mysecretpassword@mybroker:5672
Management Tool for QPID
qpid:
```

2. Check the ACL list to obtain the object ID:

```
qpid: list acl
Object Summary:
    ID   Created   Destroyed  Index
    ================================
    103  12:57:41  -          116
```

3. Optionally, you can examine the ACL:

```
qpid: show 103
Object of type: org.apache.qpid.acl:acl:_data(23510fc1-dc51-a952-39c2-
e18475c1677e)
    Attribute              103
    ================================================
    brokerRef              116
```

```
    policyFile              /tmp/reload.acl
    enforcingAcl            True
    transferAcl             False
    lastAclLoad             Tue Oct 30 12:57:41 2012
    maxConnectionsPerIp     0
    maxConnectionsPerUser   0
    maxQueuesPerUser        0
    aclDenyCount            0
    connectionDenyCount     0
    queueQuotaDenyCount     0
```

4. To reload the ACL, call the reload method of the ACL object:

```
qpid: call 103 reloadACLFile
qpid: OK (0) - {}
```

Reload ACL from program code

The broker ACL can be reloaded at runtime by calling a QMF method.

The following code calls the appropriate QMF method to reload the ACL:

Python

```
import qmf.console
qmf = qmf.console.Session()
qmf_broker = qmf.addBroker('localhost:5672')
acl = qmf.getObjects(_class="acl")[0]
result = acl.reloadACLFile()
print result
```

Note that the server must be started with ACL enabled for the reload operation to succeed.

Report a bug

8.3.5. Writing an Access Control List

1. The user id in the ACL file is of the form *<user-id>@<domain>*. The Domain is configured via the SASL configuration for the broker, and the domain/realm for qpidd is set using **`--realm`** and default to 'QPID'.

2. Each line in an ACL file grants or denies specific rights to a user.

    a. If the last line in an ACL file is **`acl deny all all`**, the ACL uses *deny mode*, and only those rights that are explicitly allowed are granted:

    ```
    acl allow user@QPID all all
    acl deny all all
    ```

    On this server, deny mode is the default. **`user@QPID`** can perform any action, but nobody else can.

    b. If the last line in an ACL file is **`acl allow all all`**, the ACL uses *allow mode*, and all rights are granted except those that are explicitly denied.

```
acl deny user@QPID all all
acl allow all all
```

On this server, allow mode is the default. The ACL allows everyone else to perform any action, but denies **user@QPID** all permissions.

3. ACL processing ends when one of the following lines is encountered:

```
acl allow all all
```

```
acl deny all all
```

Any lines after one of these statements will be ignored:

```
acl allow all all
acl deny user@QPID all all # This line is ignored !!!
```

4. ACL syntax allows fine-grained access rights for specific actions:

```
acl allow carlt@QPID create exchange name=carl.*
acl allow fred@QPID create all
acl allow all consume queue
acl allow all bind exchange
acl deny all all
```

5. An ACL file can define user groups, and assign permissions to them:

```
group admin ted@QPID martin@QPID
acl allow admin create all
acl deny all all
```

Report a bug

8.3.6. ACL Syntax

ACL rules must be on a single line and follow this syntax:

```
acl permission {<group-name>|<user-name>|"all"} {action|"all"}
[object|"all"] [property=<property-value>]
```

In ACL files, the following syntactic conventions apply:

- The default (anonymous) exchange is identified using **name=amq.default**.
- A line starting with the # character is considered a comment and is ignored.
- Empty lines and lines that contain only whitespace are ignored
- All tokens are case sensitive. ***name1*** is not the same as ***Name1*** and ***create*** is not the same as ***CREATE***
- Group lists can be extended to the following line by terminating the line with the \ character
- Additional whitespace - that is, where there is more than one whitespace character - between and after tokens is ignored. Group and ACL definitions must start with either **group** or **acl** and with no preceding

whitespace.

- All ACL rules are limited to a single line
- Rules are interpreted from the top of the file down until the name match is obtained; at which point processing stops.
- The keyword ***all*** matches all individuals, groups and actions
- The last line of the file - whether present or not - will be assumed to be **acl deny all all**. If present in the file, all lines below it are ignored.
- Names and group names may contain only ***a-z***, ***A-Z***, ***0-9***, **-** and _
- Rules must be preceded by any group definitions they can use. Any name not defined as a group will be assumed to be that of an individual.
- Qpid fails to start if ACL file is not valid
- ACL rules can be reloaded at runtime by calling a QMF method

See Also:

- Section 8.3.4, “Reloading the ACL”

Report a bug

8.3.7. ACL Definition Reference

The following tables show the possible values for **permission**, **action**, **object**, and **property** in an ACL rules file.

Table 8.3. ACL Rules: permission

allow	Allow the action
allow-log	Allow the action and log the action in the event log
deny	Deny the action
deny-log	Deny the action and log the action in the event log

Table 8.4. ACL Rules: action

consume	Applied when subscriptions are created
publish	Applied on a per message basis on publish message transfers, this rule consumes the most resources
create	Applied when an object is created, such as bindings, queues, exchanges, links
access	Applied when an object is read or accessed
bind	Applied when objects are bound together
unbind	Applied when objects are unbound
delete	Applied when objects are deleted
purge	Similar to delete but the action is performed on more than one object
update	Applied when an object is updated

Table 8.5. ACL Rules: object

queue	A queue
exchange	An exchange
broker	The broker
link	A federation or inter-broker link
method	Management or agent or broker method

Table 8.6. ACL Rules: property

name	String. Object name, such as a queue name or exchange name.
durable	Boolean. Indicates the object is durable
routingkey	String. Specifies routing key
autodelete	Boolean. Indicates whether or not the object gets deleted when the connection is closed
exclusive	Boolean. Indicates the presence of an ***exclusive*** flag
type	String. Type of object, such as topic, fanout, or xml
alternate	String. Name of the alternate exchange
queuename	String. Name of the queue (used only when the object is something other than ***queue***
schemapackage	String. QMF schema package name
schemaclass	String. QMF schema class name
policytype	String. The limit policy for a queue. Only used in rules for queue creation.
maxqueuesize	Integer. The largest value of the maximum queue size (in bytes) with which a queue is allowed to be created. Only used in rules for queue creation.
maxqueuecount	Integer. The largest value of the maximum queue depth (in messages) that a queue is allowed to be created. Only used in rules for queue creation.

Report a bug

8.3.8. Enforcing Queue Size Limits via ACL

The maximum queue size can be enforced via an ACL. This allows the administrator to disallow users from creating queues that could consume too many system resources.

CREATE QUEUE rules have ACL rules that limit the upper and lower bounds of both in-memory queue and on-disk queue store sizes.

Table 8.7. Queue Size ACL Rules

User Option	ACL Limit Property	Units
qpid.max_size	queuemaxsizelowerlimit	bytes
	queuemaxsizeupperlimit	bytes
qpid.max_count	queuemaxcountlowerlimit	messages
	queuemaxcountupperlimit	messages
qpid.max_pages_loaded	pageslowerlimit	pages
	pagesupperlimit	pages

User Option	ACL Limit Property	Units
qpid.page_factor	pagefactorlowerlimit	integer (multiple of the platform-defined page size)
	pagefactorupperlimit	integer (multiple of the platform-defined page size)

ACL Limit Properties are evaluated when the user presents one of the options in a CREATE QUEUE request. If the user's option is not within the limit properties for an ACL Rule that would allow the request, then the rule is matched with a Deny result.

Limit properties are ignored for Deny rules.

Example:

```
# Example of ACL specifying queue size constraints
# Note: for legibility this acl line has been split into multiple lines.
acl allow bob@QPID create queue name=q6 queuemaxsizelowerlimit=500000
                                        queuemaxsizeupperlimit=1000000
                                        queuemaxcountlowerlimit=200
                                        queuemaxcountupperlimit=300
```

These limits come into play when a queue is created as illustrated here:

C++

```
int main(int argc, char** argv) {
const char* url = argc>1 ? argv[1] : "amqp:tcp:127.0.0.1:5672";
const char* address = argc>2 ? argv[2] :
    "message_queue; “
    “ { create: always, “
    “   node: “
    “   { type: queue, “
    “     x-declare: ”
    “     { arguments: “
    “       { qpid.max_count:101,”
    “         qpid.max_size:1000000”
    “       }”
    “     }”
    “   }”
    “ }";
std::string connectionOptions = argc > 3 ? argv[3] : "";

Connection connection(url, connectionOptions);
try {
    connection.open();
    Session session = connection.createSession();
    Sender sender = session.createSender(address);
...
```

This queue can also be created with the **qpid-config** command:

```
qpid-config add queue --max-queue-size=1000000 --max-queue-count=101
```

When the ACL rule is processed assume that the actor, action, object, and object name all match and so this allow rule matches for the allow or deny decision. However, the ACL rule is further constrained to limit 500000 <= max_size <= 1000000 and 200 <= max_count <= 300. Since the **`queue_option max_count`** is 101 then the size limit is violated (it is too low) and the allow rule is returned with a deny decision.

Note that it is not mandatory to set *both* an upper limit *and* a lower limit. It is possible to set only a lower limit, or only an upper limit.

Report a bug

8.3.9. Resource Quota Options

The maximum number of connections can be restricted with the **`--max-connections`** broker option.

Table 8.8. Resource Quota Options

Option	Description	Default Value
`--max-connections N`	Total concurrent connections to the broker.	500
`--max-negotiate-time N`	The time during which initial protocol negotiation must succeed. This prevents resource starvation by badly behaved clients or transient network issues that prevent connections from completing.	500

Notes

- **`--max-connections`** is a qpid core limit and is enforced whether ACL is enabled or not.
- **`--max-connections`** is enforced per Broker. In a cluster of *N* nodes where all Brokers set the maximum connections to 20 the total number of allowed connections for the cluster will be *N**20.

ACL-based Quotas

To enable ACL-based quotas, an ACL file must be loaded:

Table 8.9. ACL Command-line Option

Option	Description	Default Value
`--acl-file FILE (policy.acl)`	The policy file to load from, loaded from data dir.	

When an ACL file is loaded, the following ACL options can be specified at the command-line to enforce resource quotas:

Table 8.10. ACL-based Resource Quota Options

Option	Description	Default Value
`--connection-limit-per-user N`	The maximum number of connections allowed per user. 0 implies no limit.	0

Option	Description	Default Value
--connection-limit-per-ip *N*	The maximum number of connections allowed per host IP address. 0 implies no limit.	0
--max-queues-per-user *N*	Total concurrent queues created by individual user	0

Notes

- In a cluster system the actual number of connections may exceed the connection quota value **N** by one less than the number of member nodes in the cluster. For example: in a 5-node cluster, with a limit of 20 connections, the actual number of connections can reach 24 before limiting takes place.
- Cluster connections are checked against the connection limit when they are established. The cluster connection is denied if a free connection is not available. After establishment, however, a cluster connection does not consume a connection.
- Allowed values for **N** are 0..65535.
- These limits are enforced per *cluster*.
- A value of zero (0) disables that option's limit checking.
- Per-user connections are identified by the authenticated user name.
- Per-ip connections are identified by the **<broker-ip><broker-port>-<client-ip><client-port>** tuple which is also the management connection index.
  - With this scheme host systems may be identified by several names such as **localhost** IPv4, **127.0.0.1** IPv4, or **::1** IPv6, and a separate set of connections is allowed for each name.
  - Per-IP connections are counted regardless of the user credentials provided with the connections. An individual user may be allowed 20 connections but if the client host has a 5 connection limit then that user may connect from that system only 5 times.

Report a bug

8.3.10. Per-user Resource Quotas

Resource quotas may be set on a per-user basis using ACL for fine-grained control.

Rule Syntax

The per-user ACL rule syntax is:

```
 quota connections|queues value <group-name-list>|<user-name-list> [ <group-name-list>|<user-name-list>]
```

Connection quotas

Connection quotas work in conjunction with the command line switch '**--connection-limit-per-user N**' to limit users to some number of concurrent connections.

- If the command line switch '**--connection-limit-per-user**' is absent and there are no 'quota connections' rules in the ACL file then connection limits are not enforced.

- If the command line switch '**--connection-limit-per-user**' is present then it assigns an initial value for the pseudo-user 'all'.
- If the ACL file specifies a quota for pseudo user 'all' than that value is applied to all users who are otherwise unnamed in the ACL file.
- Connection quotas for users are registered in order as the rule file is processed. A user may be assigned any number of connection quota values but only the final value is retained and enforced.
- Connection quotas for groups are applied as connection quotas for each individual user in the group at the time the 'quota connections' line is processed.
- Quota values range from 0 to 65530. A value of zero (0) denies connections.

Queue quota

Queue quotas work in conjunction with the command line switch '**--max-queues-per-user N**' to limit users to some number of concurrent queues.

- If the command line switch '**--max-queues-per-user**' is absent and there are no 'quota queues' rules in the ACL file then queue limits are not enforced.
- If the command line switch '**--max-queues-per-user**' is present then it assigns an initial value for the pseudo-user 'all'.
- If the ACL file specifies a quota for pseudo user 'all' than that value is applied to all users who are otherwise unnamed in the ACL file.
- Queue quotas for users are registered in order as the rule file is processed. A user may be assigned any number of queue quota values but only the final value is retained and enforced.
- Queue quotas for groups are applied as queue quotas for each individual user in the group at the time the 'quota queues' line is processed.
- Quota values range from 0 to 65530. A value of zero (0) denies queue creation actions.

Report a bug

8.3.11. Connection Limits by Hostname

The 0.30 C++ Broker ACL module adds the ability to create allow and deny lists of the TCP/IP hosts from which users may connect. The rule accepts these forms:

```
acl allow user create connection host=host1
acl allow user create connection host=host1,host2
acl deny  user create connection host=all
```

Using the form **host=host1** specifies a single host. With a single host the name may resolve to multiple TCP/IP addresses. For example *localhost* resolves to both *127.0.0.1* and *::1* and possibly many other addresses. A connection from any of the addresses associated with this host matches the rule and the connection is allowed or denied accordingly.

Using the form **host=host1,host2** specifies a range of TCP/IP addresses. With a host range each host must resolve to a single TCP/IP address and the second address must be numerically larger than the first. A connection from any host where host >= host1 and host <= host2 match the rule and the connection is allowed or denied accordingly.

Using the form **`host=all`** specifies all TCP/IP addresses. A connection from any host matches the rule and the connection is allowed or denied accordingly.

Connection denial is only applied to incoming TCP/IP connections. Other socket types are not subjected to nor denied by range checks.

Connection creation rules are divided into three categories:

1. User = all, host != all

   These define global rules and are applied before any specific user rules. These rules may be used to reject connections before any AMPQ protocol is run and before any user names have been negotiated.

2. User != all, host = any legal host or 'all'

   These define user rules. These rules are applied after the global rules and after the AMQP protocol has negotiated user identities.

3. User = all, host = all

   This rule defines what to do if no other rule matches. The default value is "ALLOW". Only one rule of this type may be defined.

The following example illustrates how this feature can be used:

Example 8.1. Connection Limits by Host Name

```
group admins alice bob chuck
group Company1 c1_usera c1_userb
group Company2 c2_userx c2_usery c2_userz
acl allow admins   create connection host=localhost
acl allow admins   create connection host=10.0.0.0,10.255.255.255
acl allow admins   create connection host=192.168.0.0,192.168.255.255
acl allow admins   create connection host=[fc00::],[fc00::ff]
acl allow Company1 create connection host=company1.com
acl deny  Company1 create connection host=all
acl allow Company2 create connection host=company2.com
acl deny  Company2 create connection host=all
```

In this example admins may connect from localhost or from any system on the 10.0.0.0/24, 192.168.0.0/16, and fc00::/7 subnets. Company1 users may connect only from company1.com and Company2 users may connect only from company2.com. However, this example has a flaw. Although the admins group has specific hosts from which it is allowed to make connections it is not blocked from connecting from anywhere. The Company1 and Company2 groups are blocked appropriately. This ACL file may be rewritten as follows:

```
group admins alice bob chuck
group Company1 c1_usera c1_userb
group Company2 c2_userx c2_usery c2_userz
acl allow admins   create connection host=localhost
acl allow admins   create connection host=10.0.0.0,10.255.255.255
acl allow admins   create connection host=192.168.0.0,192.168.255.255
acl allow admins   create connection host=[fc00::],[fc00::ff]
acl allow Company1 create connection host=company1.com
acl allow Company2 create connection host=company2.com
acl deny  all      create connection host=all
```

Now the listed administrators are blocked from connecting from anywhere but their allowed hosts.

Report a bug

8.3.12. Routing Key Wildcards

Topic Exchange match logic is used for ACL rules containing a **routingkey** property. These rules include:

- bind exchange <name> routingkey=*X*
- unbind exchange <name> routingkey=*X*
- publish exchange <name> routingkey=*X*

The **routingkey** property is now matched using the same logic as the Topic Exchange match. This allows administrators to express user limits in flexible terms that map to the namespace where **routingkey** values are used.

Wildcard matching and Topic Exchanges

In the binding key, # matches any number of period-separated terms, and * matches a single term.

So a binding key of **#.news** will match messages with subjects such as **usa.news** and **germany.europe.news**, while a binding key of ***.news** will match messages with the subject **usa.news**, but not **germany.europe.news**.

Example:

The following ACL rules:

```
acl allow-log uHash1@COMPANY publish exchange name=X routingkey=a.#.b
acl deny all all
```

Produce the following results when user **uHash1@COMPANY** publishes to exchange X:

Table 8.11.

routingkey in publish to exchange X	result
a.b	allow-log
a.x.b	allow-log
a..x.y.zz.b	allow-log
a.b.	deny
q.x.b	deny

Report a bug

8.3.13. Routing Key Wildcard Examples

The following table demonstrates the interaction between routing key wildcards in ACL rules and message headers. Along the top of the table are routing keys with wildcards. Down the left-hand side are message headers.

The character **X** indicates that the message with the given header will *not* be routed given an ACL rule that allows messages with the specified routing key. 'Routed' indicates that the message with the given header *will* be routed by an ACL rule that allows message with the specified routing key.

Table 8.12. Routing Keys, Message Headers, and Resultant Routing.

Routing Keys ->	a.#	#.e	a.#.e	a.#.c.#.e	#.c.#	#
Message Headers:						
ax	X	X	X	X	X	Routed
a.x	Routed	X	X	X	X	Routed
ex	X	X	X	X	X	Routed
e.x	X	X	X	X	X	Routed
ae	X	X	X	X	X	Routed
a.e	Routed	Routed	Routed	X	X	Routed
a..e	Routed	Routed	Routed	X	X	Routed
a.x.e	Routed	Routed	Routed	X	X	Routed
a.c.e	Routed	Routed	Routed	Routed	Routed	Routed
a..c..e	Routed	Routed	Routed	Routed	Routed	Routed
a.b.c.d.e	Routed	Routed	Routed	Routed	Routed	Routed
a.b.x.c.d.y.e	Routed	Routed	Routed	Routed	Routed	Routed
a.#	Routed	X	X	X	X	Routed
#.e	X	Routed	X	X	X	Routed
a.#.e	Routed	Routed	Routed	X	X	Routed
a.#.c.#.e	Routed	Routed	Routed	Routed	Routed	Routed
#.c.#	X	X	X	X	Routed	Routed
#	X	X	X	X	X	Routed

Report a bug

8.3.14. User Name and Domain Name Symbol Substitution

MRG 3 has the ability to use a simple set of user name and domain name substitution variables. This provides administrators with an easy way to define private or shared resources.

Symbol substitution is allowed in the Acl file anywhere that text is supplied for a property value.

In the following table an authenticated user **bob.user@QPID.COM** has his substitution keywords expanded.

Table 8.13.

Keyword	Expansion
${userdomain}	bob_user_QPID_COM
${user}	bob_user
${domain}	QPID_COM

The original name has the period "." and at symbol "@" characters translated into underscore "_". This allows substitutions to work when the substitution keyword is used in a routingkey in the ACL file.

Using Symbol Substitution and Wildcards in Routing Keys

The ***** symbol can be used a wildcard match for any number of characters in a single field in a routing key. For example:

```
acl allow user_group publish exchange name=users routingkey=${user}-
delivery-*
```

The '#' symbol , when used in a routing key specification substitutes for any number of dotted subject name fields. User and Domain symbol substitutions can also be combined with the # wildcard symbol in routing keys, for example:

```
acl allow user_group bind exchange name=${user}-work2
routingkey=news.#.${user}
```

ACL Matching of Wildcards in Routing Keys

The ACL processing matches **${userdomain}** before matching either **${user}** or **${domain}**. In most circumstances ACL processing treats **${user}_${domain}** and **${userdomain}** as equivalent and the two forms may be used interchangeably. The exception to this is rules that specify wildcards within routing keys. In this case the combination **${user}_${domain}** will never match, and the form **${userdomain}** should be used.

For example, the following rule will never match:

```
acl allow all publish exchange name=X routingkey=${user}_${domain}.c
```

In that example, the rule will never match, as the ACL processor looks for routingkey **${userdomain}.c**.

ACL Symbol Substitution Example

Administrators can set up ACL rule files that allow every user to create a private exchange, a private queue, and a private binding between them. In this example the users are also allowed to create private backup exchanges, queues and bindings. This effectively provides limits to user's exchange, queue, and binding creation and guarantees that each user gets exclusive access to these resources.

```
#
# Create primary queue and exchange:
acl allow all create  queue    name=${user}-work alternate=${user}-work2
acl deny  all create  queue    name=${user}-work alternate=*
acl allow all create  queue    name=${user}-work
acl allow all create  exchange name=${user}-work alternate=${user}-work2
acl deny  all create  exchange name=${user}-work alternate=*
acl allow all create  exchange name=${user}-work
#
# Create backup queue and exchange
#
acl deny  all create  queue    name=${user}-work2 alternate=*
acl allow all create  queue    name=${user}-work2
acl deny  all create  exchange name=${user}-work2 alternate=*
acl allow all create  exchange name=${user}-work2
#
# Bind/unbind primary exchange
#
```

```
acl allow all bind   exchange name=${user}-work routingkey=${user}
queuename=${user}-work
acl allow all unbind exchange name=${user}-work routingkey=${user}
queuename=${user}-work
#
# Bind/unbind backup exchange
#
acl allow all bind   exchange name=${user}-work2 routingkey=${user}
queuename=${user}-work2
acl allow all unbind exchange name=${user}-work2 routingkey=${user}
queuename=${user}-work2
#

# deny mode
#
acl deny all all
```

Report a bug

8.3.15. ACL Definition Examples

Most ACL files begin by defining groups:

```
group admin ted@QPID martin@QPID
group user-consume martin@QPID ted@QPID
group group2 kim@QPID user-consume rob@QPID
group publisher group2 \
tom@QPID andrew@QPID debbie@QPID
```

Rules in an ACL file grant or deny specific permissions to users or groups:

```
acl allow carlt@QPID create exchange name=carl.*
acl allow rob@QPID create queue
acl allow guest@QPID bind exchange name=amq.topic routingkey=stocks.rht.#
acl allow user-consume create queue name=tmp.*

acl allow publisher publish all durable=false
acl allow publisher create queue name=RequestQueue
acl allow consumer consume queue durable=true
acl allow fred@QPID create all
acl allow bob@QPID all queue
acl allow admin all
acl allow all consume queue
acl allow all bind exchange
acl deny all all
```

In the previous example, the last line, **`acl deny all all`**, denies all authorizations that have not been specifically granted. This is the default, but it is useful to include it explicitly on the last line for the sake of clarity. If you want to grant all rights by default, you can specify **`acl allow all all`** in the last line.

Do not allow ***`guest`*** to access and log QMF management methods that could cause security breaches:

```
group allUsers guest@QPID
....
acl deny-log allUsers create link
```

```
acl deny-log allUsers access method name=connect
acl deny-log allUsers access method name=echo
acl allow all all
```

Report a bug

Chapter 9. High Availability

9.1. Clustering (High Availability)

9.1.1. Changes to Clustering in MRG 3

MRG 3 replaces the **`cluster`** module with the new **ha** module. This module provides active-passive clustering functionality for high availability.

The **`cluster`** module in MRG 2 was active-active: clients could connect to any broker in the cluster. The new **ha** module is active-passive. Exactly one broker acts as *primary* the other brokers act as *backup*. Only the primary accepts client connections. If a client attempts to connect to a backup broker, the connection is aborted and the client fails-over until it connects to the primary.

The new **ha** module also supports a *virtual IP address*. Clients can be configured with a single IP address that is automatically routed to the primary broker. This is the recommended configuration.

The fail-over exchange is provided for backwards compatibility. New implementations should use a virtual IP address instead.

Improvement to multi-threaded performance

In MRG 2, a clustered broker would only utilize a single CPU thread. Some users worked around this by running multiple clustered brokers on a single machine, to utilize the multiple cores.

In MRG 3, a clustered broker now utilizes multiple threads and can take advantage of multi-core CPUs.

Report a bug

9.1.2. Active-Passive Messaging Clusters

The High Availability (HA) module provides *active-passive*, *hot-standby* messaging clusters to provide fault tolerant message delivery.

In an active-passive cluster only one broker, known as the primary, is active and serving clients at a time. The other brokers are standing by as backups. Changes on the primary are replicated to all the backups so they are always up-to-date or "hot". Backup brokers reject client connection attempts, to enforce the requirement that clients only connect to the primary.

If the primary fails, one of the backups is promoted to take over as the new primary. Clients fail-over to the new primary automatically. If there are multiple backups, the other backups also fail-over to become backups of the new primary.

This approach relies on an external cluster resource manager, **`rgmanager`**, to detect failures, choose the new primary and handle network partitions.

Report a bug

9.1.3. Avoiding Message Loss

To avoid message loss, the primary broker sends acknowledgment of messages received from clients when the message has been replicated and acknowledged by all of the back-up brokers, or has been consumed from the primary queue.

This ensures that all acknowledged messages are safe: they have either been consumed or backed up to all backup brokers. Messages that are consumed before they are replicated do not need to be replicated. This reduces the work load when replicating a queue with active consumers.

Clients keep unacknowledged messages in the client replay buffer until they are acknowledged by the primary. If the primary fails, clients will fail-over to the new primary and re-send all their unacknowledged messages.

If the primary crashes, all the acknowledged messages will be available on the backup that takes over as the new primary. The unacknowledged messages will be re-sent by the clients. Thus no messages are lost.

Note that this means it is possible for messages to be duplicated. In the event of a failure it is possible for a message to be received by the backup that becomes the new primary and re-sent by the client. The application must take steps to identify and eliminate duplicates.

When a new primary is promoted after a fail-over it is initially in "recovering" mode. In this mode, it delays acknowledgment of messages on behalf of all the backups that were connected to the previous primary. This protects those messages against a failure of the new primary until the backups have a chance to connect and catch up.

Not all messages need to be replicated to the back-up brokers. If a message is consumed and acknowledged by a regular client before it has been replicated to a backup, then it doesn't need to be replicated.

Report a bug

9.1.4. HA Broker States

Joining

Initial status of a new broker that has not yet connected to the primary.

Catch-up

A backup broker that is connected to the primary and catching up on queues and messages.

Ready

A backup broker that is fully caught-up and ready to take over as primary.

Recovering

The newly-promoted primary, waiting for backups to connect and catch up.

Active

The active primary broker with all backups connected and caught-up.

Report a bug

9.1.5. Limitations in HA in MRG 3

There are some limitations to HA support in MRG 3:

- HA replication is limited to 65434 queues.
- Manual reallocation of **`qpidd-primary`** service cannot be done to a node where the qpid broker is not in ready state (is stopped, or either in catchup or joining state).

- Failback with cluster ordered failover-domains ('**ordered=1**' in **cluster.conf**) can cause an infinite failover loop under certain conditions. To avoid this, use cluster ordered failover-domains with **nofailback=1** specified in **cluster.conf**.
- Local transactional changes are replicated atomically. If the primary crashes during a local transaction, no data is lost. Distributed transactions are not yet supported by HA Cluster.
- Configuration changes (creating or deleting queues, exchanges and bindings) are replicated asynchronously. Management tools used to make changes will consider the change complete when it is complete on the primary, however it may not yet be replicated to all the backups.
- Federation links to the primary will not fail over correctly. Federated links from the primary will be lost in fail over, they will not be re-connected to the new primary. It is possible to work around this by replacing the qpidd-primary start up script with a script that re-creates federation links when the primary is promoted.

Report a bug

9.1.6. Broker HA Options

Options for the qpid-ha Broker Utility

ha-cluster *yes|no*

Set to "yes" to have the broker join a cluster.

ha-queue-replication *yes|no*

Enable replication of specific queues without joining a cluster.

ha-brokers-url *URL*

The URL used by cluster brokers to connect to each other. The URL must contain a comma separated list of the broker addresses, rather than a virtual IP address.

The full format of the URL is given by this grammar:

```
url = ["amqp:"][ user ["/" password] "@" ] addr ("," addr)*
addr = tcp_addr / rmda_addr / ssl_addr / ...
tcp_addr = ["tcp:"] host [":" port]
rdma_addr = "rdma:" host [":" port]
ssl_addr = "ssl:" host [":" port]'>
```

ha-public-url *URL*

This option is only needed for backwards compatibility if you have been using the **amq.failover** exchange. This exchange is now obsolete, it is recommended to use a virtual IP address instead.

If set, this URL is advertized by the **amq.failover** exchange and overrides the broker option **known-hosts-url**.

ha-replicate *VALUE*

Specifies whether queues and exchanges are replicated by default. *VALUE* is one of: **none**, **configuration**, **all**.

ha-username *USER*, ha-password *PASS*, ha-mechanism *MECHANISM*

Authentication settings used by HA brokers to connect to each other. If you are using authorization

Authentication settings used by HA brokers to connect to each other. If you are using authentication then this user must have all permissions.

ha-backup-timeout *SECONDS*

Maximum time that a recovering primary will wait for an expected backup to connect and become ready.

Values specified as *SECONDS* can be a fraction of a second, e.g. "0.1" for a tenth of a second. They can also have an explicit unit, e.g. 10s (seconds), 10ms (milliseconds), 10us (microseconds), 10ns (nanoseconds)

link-maintenance-interval *SECONDS*

HA uses federation links to connect from backup to primary. Backup brokers check the link to the primary on this interval and re-connect if need be. Default 2 seconds. Can be set lower for faster failover (e.g. 0.1 seconds). Setting too low will result in excessive link-checking on the backups.

link-heartbeat-interval *SECONDS*

The number of seconds to wait for a federated link heart beat or the timeout for broker status checks.

By default this is 120 seconds. Provide a lower value (for example, 10 seconds) to enable faster failover detection in a HA scenario. If the value is set too low, a slow broker may be considered as failed and will be killed.

If no heartbeat is received for twice this interval the primary will consider that backup dead (e.g. if backup is hung or partitioned.)

It may take up to this interval for rgmanager to detect a hung or partitioned broker. The primary may take up to twice this interval to detect a hung or partitioned backup. Clients sending messages may also be delayed during this time.

To configure a HA cluster you must set at least **`ha-cluster`** and **`ha-brokers-url`**.

See Also:

- Section 9.1.17, “Controlling replication of queues and exchanges”

Report a bug

9.1.7. Firewall Configuration for Clustering

The following ports are used on a clustered system, and must be opened on the firewall:

Table 9.1. Ports Used by Clustered Systems

Port	Protocol	Component
5404	UDP	cman
5405	UDP	cman
5405	TCP	luci
8084	TCP	luci
11111	TCP	ricci
14567	TCP	gnbd
16851	TCP	modclusterd
21064	TCP	dlm

Port	Protocol	Component
50006	TCP	ccsd
50007	UDP	ccsd
50008	TCP	ccsd
50009	TCP	ccsd

The following **iptables** commands, when run with root privileges, will configure the system to allow communication on these ports.

```
iptables -I INPUT -p udp -m udp --dport 5405  -j ACCEPT
iptables -I INPUT -p tcp -m tcp --dport 5405  -j ACCEPT
iptables -I INPUT -p tcp -m tcp --dport 8084  -j ACCEPT
iptables -I INPUT -p tcp -m tcp --dport 11111  -j ACCEPT
iptables -I INPUT -p tcp -m tcp --dport 14567  -j ACCEPT
iptables -I INPUT -p tcp -m tcp --dport 16851  -j ACCEPT
iptables -I INPUT -p tcp -m tcp --dport 21064  -j ACCEPT
iptables -I INPUT -p tcp -m tcp --dport 50006  -j ACCEPT
iptables -I INPUT -p udp -m udp --dport 50007  -j ACCEPT
iptables -I INPUT -p tcp -m tcp --dport 50008  -j ACCEPT
iptables -I INPUT -p tcp -m tcp --dport 50009  -j ACCEPT
service iptables save
service iptables restart
```

Report a bug

9.1.8. ACL Requirements for Clustering

Clustering requires federation links between brokers. When the broker has **auth=yes**, all federation links are disallowed by default. The following ACL rule is required to allow the federation used by HA Clustering:

```
acl allow <ha-username> all all
```

Report a bug

9.1.9. Cluster Resource Manager (rgmanager)

Broker fail-over is managed by the resource manager **rgmanager**.

The resource manager is responsible for starting the **qpidd** broker on each node in the cluster. The resource manager then promotes one of the brokers to be the primary. The other brokers connect to the primary as backups, using the URL provided in the **ha-brokers-url** configuration option.

Once connected, the backup brokers synchronize their state with the primary. When a backup is synchronized, or "hot", it is ready to take over if the primary fails. Backup brokers continually receive updates from the primary in order to stay synchronized.

If the primary fails, backup brokers go into fail-over mode. The resource manager detects the failure and promotes one of the backups to be the new primary. The other backups connect to the new primary and synchronize their state with it.

The resource manager also protects the cluster from split-brain conditions resulting from a network partition. A network partition divides a cluster into two sub-groups which cannot see each other. A quorum voting algorithm disables nodes in the inquorate sub-group.

Report a bug

9.1.10. Install HA Cluster Components

Procedure 9.1. Qpidd HA Component Installation Steps

1. Open a terminal and switch to the superuser account.
2. Run **`yum install qpid-cpp-server-ha`** to install all required components.

Procedure 9.2. Red Hat Linux HA Cluster Components Installation Steps

1. Subscribe the system to the "RHEL Server High Availability" channel.
2. Open a terminal and switch to the superuser account.
3. Run **`yum install -y rgmanager ccs`** to install all required components.
4. Disable the Network Manager before starting HA Clustering. HA Clustering will not work correctly with Network Manager started or enabled

```
# chkconfig NetworkManager off
```

5. Activate rgmanager, cman and ricci services.

```
# chkconfig rgmanager on
# chkconfig cman on
# chkconfig ricci on
```

6. Deactivate the qpidd service.

```
# chkconfig qpidd off
```

The **`qpidd`** service must be *off* in **`chkconfig`** because rgmanager will start and stop qpidd. If the normal system init process also attempts to start and stop qpidd it can cause rgmanager to lose track of qpidd processes.

If qpidd is not turned off, **`clustat`** shows a qpidd service to be stopped when in fact there is a qpidd process running. In this situation, the qpidd log shows errors similar to this:

```
critical Unexpected error: Daemon startup failed: Cannot lock
/var/lib/qpidd/lock: Resource temporarily unavailable
```

Report a bug

9.1.11. Virtual IP Addresses

Qpid HA Clustering supports virtual IP addresses. A virtual IP address is an IP address that is relocated to the primary node in the cluster whenever a promotion occurs. The resource manager associates this address with the primary node in the cluster, and relocates it to the new primary when there is a failure. This simplifies configuration as you can publish a single IP address rather than a list.

The Virtual IP address must be correctly configured on each node. The cluster manager's only job must be to start and stop the Virtual IP address. If you use a Virtual IP address on the same network as the node's physical adapter, then the cluster manager can bind the address to the adapter when the node is promoted to

primary, and no further configuration is required. If the Virtual IP address is on a different network than the physical adapter, then a second physical adapter or a virtual network interface must be configured for the Virtual IP.

See Also:

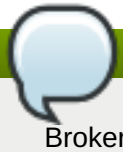
- Section 9.1.12, “Configure HA Cluster”

Report a bug

9.1.12. Configure HA Cluster

MRG Messaging brokers can be clustered using **cman** and **rgmanager** to create an active-passive, hot-standby qpidd HA cluster. For further information on the underlying clustering technologies **cman** and **rgmanager**, refer to the *Red Hat Enterprise Linux Cluster Administration Guide*.

HA Clustering uses the **/etc/cluster/cluster.conf** file to configure **cman** and **rgmanager**.



Note

Broker management is required for HA to operate. It is enabled by default. The option **mgmt-enable** must not be set to "no".

Note

Incorrect security settings are a common cause of problems when getting started, see Section 9.1.19, “Security”.

The tool **ccs** provides a high-level user-friendly mechanism to configure the **cluster.conf** file, and is the recommended method for configuring a cluster. Refer to the *Red Hat Enterprise Linux Cluster Administration Guide* for more information on using the **ccs** tool.

The following steps use **ccs** to create an example cluster of 3 nodes named **node1**, **node2** and **node3**. Run the following as the **root** user:

1. Start the **ricci** service:

```
service ricci start
```

2. If you have not previously set the **ricci** password, set it now:

```
passwd ricci
```

3. Create a new cluster:

```
ccs -h localhost --createcluster qpid-test
```

4. Add three nodes:

```
ccs -h localhost --addnode node1.example.com
ccs -h localhost --addnode node2.example.com
ccs -h localhost --addnode node3.example.com
```

5. Add a **failoverdomain** for each:

```
ccs -h localhost --addfailoverdomain node1-domain restricted
ccs -h localhost --addfailoverdomain node2-domain restricted
ccs -h localhost --addfailoverdomain node3-domain restricted
```

6. Add a **failoverdomainnode** for each:

```
ccs -h localhost --addfailoverdomainnode node1-domain
node1.example.com
ccs -h localhost --addfailoverdomainnode node2-domain
node2.example.com
ccs -h localhost --addfailoverdomainnode node3-domain
node3.example.com
```

7. Add the scripts:

```
ccs -h localhost --addresource script name=qpidd
file=/etc/init.d/qpidd
ccs -h localhost --addresource script name=qpidd-primary
file=/etc/init.d/qpidd-primary
```

8. Add the Virtual IP Address:

```
ccs -h localhost --addresource ip address=20.0.20.200 monitor_link=1
```

9. Add the **qpidd** service for each node. It should be restarted if it fails:

```
ccs -h host --addservice node1-qpidd-service domain=node1-domain
recovery=restart
ccs -h localhost --addsubservice node1-qpidd-service script ref=qpidd
ccs -h localhost --addservice node2-qpidd-service domain=node2-domain
recovery=restart
ccs -h localhost --addsubservice node2-qpidd-service script ref=qpidd
ccs -h localhost --addservice node3-qpidd-service domain=node3-domain
recovery=restart
ccs -h localhost --addsubservice node3-qpidd-service script ref=qpidd
```

10. Add the primary **qpidd** service. It only runs on a single node at a time, and can run on any node:

```
ccs --host localhost --addservice qpidd-primary-service
recovery=relocate autostart=1 exclusive=0
ccs -h localhost --addsubservice qpidd-primary-service script
ref=qpidd-primary
ccs -h localhost --addsubservice qpidd-primary-service ip
ref=20.0.20.200
```

Here is a commented version of the **/etc/cluster/cluster.conf** file produced by the previous steps:

```
<?xml version="1.0"?>
<!--
This is an example of a cluster.conf file to run qpidd HA under rgmanager.
This example configures a 3 node cluster, with nodes named node1, node2 and
node3.

NOTE: fencing is not shown, you must configure fencing appropriately for your
cluster.
-->

<cluster name="qpid-test" config_version="18">
    <!-- The cluster has 3 nodes. Each has a unique nodeid and one vote
         for quorum. -->
    <clusternodes>
        <clusternode name="node1.example.com" nodeid="1"/>
        <clusternode name="node2.example.com" nodeid="2"/>
        <clusternode name="node3.example.com" nodeid="3"/>
    </clusternodes>

    <!-- Resouce Manager configuration. -->
    <rm>
        <!--
     There is a failoverdomain for each node containing just that node.
     This specifies that the qpidd service should always run on each node.
         -->

        <failoverdomains>
            <failoverdomain name="node1-domain" restricted="1">
                <failoverdomainnode name="node1.example.com"/>
            </failoverdomain>
            <failoverdomain name="node2-domain" restricted="1">
                <failoverdomainnode name="node2.example.com"/>
            </failoverdomain>
            <failoverdomain name="node3-domain" restricted="1">
                <failoverdomainnode name="node3.example.com"/>
            </failoverdomain>
        </failoverdomains>

        <resources>
            <!-- This script starts a qpidd broker acting as a backup. -->
            <script file="/etc/init.d/qpidd" name="qpidd"/>

            <!-- This script promotes the qpidd broker on this node to
primary. -->
            <script file="/etc/init.d/qpidd-primary" name="qpidd-primary"/>

            <!-- This is a virtual IP address on a seprate network for
client traffic. -->
            <ip address="20.0.20.200" monitor_link="1"/>
        </resources>

        <!-- There is a qpidd service on each node,
             it should be restarted if it fails. -->

        <service name="node1-qpidd-service" domain="node1-domain"
recovery="restart">
```

```
            <script ref="qpidd"/>
        </service>
        <service name="node2-qpidd-service" domain="node2-domain"
recovery="restart">
            <script ref="qpidd"/>
        </service>
        <service name="node3-qpidd-service" domain="node3-domain"
recovery="restart">
            <script ref="qpidd"/>
        </service>

        <!-- There should always be a single qpidd-primary service,
              it can run on any node. -->

        <service name="qpidd-primary-service" autostart="1" exclusive="0"
recovery="relocate">

            <script ref="qpidd-primary"/>
            <!-- The primary has the IP addresses for brokers and clients to
connect. -->
            <ip ref="20.0.20.200"/>
        </service>
    </rm>
</cluster>
```

There is a **failoverdomain** for each node containing just that one node. This specifies that the qpidd service always runs on all nodes.

The **resources** section defines the **qpidd** script used to start the **qpidd** service. It also defines the **qpidd-primary** script which does not actually start a new service, rather it promotes the existing **qpidd** broker to primary status. The **qpidd-primary** script is installed by the **qpid-cpp-server-ha** package.

The resources section defines the virtual IP address for client-to-broker communication.

To take advantage of the virtual IP address, **qpidd.conf** should contain these lines:

```
ha-cluster = yes
ha-public-url = 20.0.20.200
ha-brokers-url = 20.0.10.1, 20.0.10.2, 20.0.10.3
```

This configuration specifies the actual network addresses of the three nodes (**ha-brokers-url**), and the Virtual IP address for the cluster, which clients should connect to: **20.0.10.200**.

The **service** section defines 3 **qpidd** services, one for each node. Each service is in a restricted fail-over domain containing just that node, and has the **restart** recovery policy. This means that the rgmanager will run **qpidd** on each node, restarting if it fails.

There is a single **qpidd-primary-service** using the **qpidd-primary** script. It is not restricted to a domain and has the **relocate** recovery policy. This means **rgmanager** will start **qpidd-primary** on one of the nodes when the cluster starts and will relocate it to another node if the original node fails. Running the **qpidd-primary** script does not start a new broker process, it promotes the existing broker to become the primary.

Report a bug

9.1.13. Shutting Down qpidd on a HA Node

Both the per-node **qpidd** service and the re-locatable **qpidd-primary** service are implemented by the same **qpidd** daemon.

As a result, stopping the **qpidd** service will not stop a **qpidd** daemon that is acting as primary, and stopping the **qpidd-primary** service will not stop a **qpidd** process that is acting as backup.

To shut down a node that is acting as primary, you must shut down the **qpidd** service *and* relocate the primary:

```
clusvcadm -d somenode-qpidd-service
clusvcadm -r qpidd-primary-service
```

Doing this will shut down the **qpidd** daemon on that node. It will also prevent the primary service from relocating back to the node because the **qpidd** service is no longer running on that location.

Report a bug

9.1.14. Start and Stop HA Cluster

Note that starting the cluster enables cluster service startup on reboot. Stopping the cluster disables cluster service startup on reboot.

Start and Stop the HA Cluster on a node

To start the HA Cluster on a node:

```
ccs [-h host] --start
```

To stop the HA Cluster on a node:

```
ccs [-h host] --stop
```

Start and Stop the HA Cluster on all nodes

To start the HA Cluster on all configured nodes:

```
ccs [-h host] --startall
```

Note that this also enables cluster service startup on reboot.

To stop the HA Cluster on all configured nodes:

```
ccs [-h host] --stopall
```

Report a bug

9.1.15. Configure Clustering to use a non-privileged (non-root) user

When qpidd is run as a non-root user, the configuration files need to be stored in a location that is readable and writable by that user. This can be done by modifying the start-up script for qpidd in the following manner:

```
# diff -u /etc/rc.d/init.d/qpidd /etc/rc.d/init.d/qpidd-mod
--- /etc/rc.d/init.d/qpidd.orig 2014-01-15 19:06:19.000000000 +0100
```

```
+++ /etc/rc.d/init.d/qpidd      2014-02-07 16:02:47.136001472 +0100
@@ -38,6 +38,9 @@
 prog=qpidd
 lockfile=/var/lock/subsys/$prog
 pidfile=/var/run/qpidd.pid
+
+CFG_DIR=/var/lib/qpidd
+QPIDD_OPTIONS="--config ${CFG_DIR}/qpidd.conf --client-config
${CFG_DIR}/qpidc.conf"

 # Source configuration
 if [ -f /etc/sysconfig/$prog ] ; then
```

When the patch above is applied to **/etc/rc.d/init.d/qpidd**, the configuration files for the broker are read from the **/var/lib/qpidd** directory, rather than from **/etc/qpid** as they are by default.

Report a bug

9.1.16. Broker Administration Tools and HA

Normally, clients are not allowed to connect to a backup broker. However management tools are allowed to connect to a backup brokers. If you use these tools you must not add or remove messages from replicated queues, nor create or delete replicated queues or exchanges as this will disrupt the replication process and may cause message loss.

qpid-ha allows you to view and change HA configuration settings.

The tools **qpid-config**, **qpid-route** and **qpid-stat** will connect to a backup if you pass the flag **--ha-admin** on the command line.

Report a bug

9.1.17. Controlling replication of queues and exchanges

By default, queues and exchanges are not replicated automatically. You can change the default behavior by setting the ha-replicate configuration option. It has one of the following values:

all

Replicate everything automatically: queues, exchanges, bindings and messages.

configuration

Replicate the existence of queues, exchange and bindings but don't replicate messages.

none

Don't replicate anything, this is the default.

You can over-ride the default for a particular queue or exchange by passing the argument **qpid.replicate** when creating the queue or exchange. It takes the same values as **ha-replicate**.

Bindings are automatically replicated if the queue and exchange being bound both have replication all or configuration, they are not replicated otherwise.

You can create replicated queues and exchanges with the **qpid-config** management tool like this:

```
qpid-config add queue myqueue --replicate all
```

To create replicated queues and exchanges via the client API, add a node entry to the address like this:

```
"myqueue;{create:always,node:{x-declare:{arguments:
{'qpid.replicate':all}}}}"
```

There are some built-in exchanges created automatically by the broker, these exchanges are never replicated. The built-in exchanges are the default (nameless) exchange, the AMQP standard exchanges (**`amq.direct`**, **`amq.topic`**, **`amq.fanout`** and **`amq.match`**) and the management exchanges (**`qpid.management`**, **`qmf.default.direct`** and **`qmf.default.topic`**)

Note that if you bind a replicated queue to one of these exchanges, the binding will not be replicated, so the queue will not have the binding after a fail-over.

Report a bug

9.1.18. Client Connection and Fail-over

Clients can only connect to the primary broker. Backup brokers reject any connection attempt by a client. Clients rejected by a backup broker will automatically fail-over until they connect to the primary. If **`ha-public-url`** contains multiple addresses, the client will try them all in rotation. If it is a virtual IP address the client will retry on the same address until reconnected.

Clients are configured with the URL for the cluster (details below for each type of client). There are two possibilities:

1. The URL contains a single virtual IP address that is assigned to the primary broker by the resource manager. *This is the recommended configuration.*
2. The URL contains multiple addresses, one for each broker in the cluster.

In the first case the resource manager assigns the Virtual IP address to the primary broker, so clients only need to retry on a single address. In the second case, clients will repeatedly retry each address in the URL until they successfully connect to the primary.

When the primary broker fails, clients retry all known cluster addresses until they connect to the new primary. The client re-sends any messages that were previously sent but not acknowledged by the broker at the time of the failure. Similarly messages that have been sent by the broker, but not acknowledged by the client, are re-queued.

TCP can be slow to detect connection failures. A client can configure a connection to use a heartbeat to detect connection failure, and can specify a time interval for the heartbeat. If heartbeats are in use, failures will be detected no later than twice the heartbeat interval. The following sections explain how to enable heartbeat in each client.

Note: If you are using a Virtual IP address in your cluster then fail-over occurs transparently on the server, and you treat the cluster as a single broker using the Virtual IP address. The following sections explain how to configure clients with multiple addresses, for use when the cluster does not use a Virtual IP.

Suppose your cluster has 3 nodes: **`node1`**, **`node2`** and **`node3`** all using the default AMQP port, and you are not using a virtual IP address. To connect a client you need to specify the address(es) and set the reconnect property to true. The following sub-sections show how to connect each type of client.

C++ clients

With the C++ client, you specify multiple cluster addresses in a single URL. You also need to specify the

connection option **reconnect** to be true. For example:

```
qpid::messaging::Connection c("node1,node2,node3","{reconnect:true}");
```

Heartbeats are disabled by default. You can enable them by specifying a heartbeat interval (in seconds) for the connection via the **heartbeat** option. For example:

```
qpid::messaging::Connection c("node1,node2,node3","
{reconnect:true,heartbeat:10}");
```

Python clients

With the Python client, you specify **reconnect=True** and a list of **host:port** addresses as **reconnect_urls** when calling **Connection.establish** or **Connection.open**:

```
connection = qpid.messaging.Connection.establish("node1", reconnect=True,
reconnect_urls=["node1", "node2", "node3"])
```

Heartbeats are disabled by default. You can enable them by specifying a heartbeat interval (in seconds) for the connection via the '**heartbeat**' option. For example:

```
connection = qpid.messaging.Connection.establish("node1", reconnect=True,
reconnect_urls=["node1", "node2", "node3"], heartbeat=10)
```

Java JMS Clients

In Java JMS clients, client fail-over is handled automatically if it is enabled in the connection. You can configure a connection to use fail-over using the **failover** property:

```
connectionfactory.qpidConnectionfactory = amqp://guest:guest@clientid/test?
brokerlist='tcp://localhost:5672'&failover='failover_exchange'
```

This property can take three values:

Fail-over Modes

failover_exchange

If the connection fails, fail over to any other broker in the cluster.

roundrobin

If the connection fails, fail over to one of the brokers specified in the brokerlist.

singlebroker

Fail-over is not supported; the connection is to a single broker only.

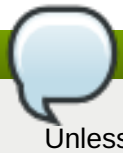
In a Connection URL, heartbeat is set using the **idle_timeout** property, which is an integer corresponding to the heartbeat period in seconds. For instance, the following line from a JNDI properties file sets the heartbeat time out to 3 seconds:

```
connectionfactory.qpidConnectionfactory = amqp://guest:guest@clientid/test?
brokerlist='tcp://localhost:5672',idle_timeout=3
```

Report a bug

9.1.19. Security

This section outlines the HA specific aspects of security configuration. Please see for more details on enabling authentication and setting up Access Control Lists.



Note

Unless you disable authentication with **auth=no** in your configuration, you *must* set the options below and you *must* have an ACL file with at least the entry described below.

Backups will be *unable to connect to the primary* if the security configuration is incorrect. See also Section 9.3.1, “Troubleshooting Cluster configuration”

When authentication is enabled you must set the credentials used by HA brokers with following options:

Table 9.2. HA Security Options

HA Security Options	
ha-username *USER*	User name for HA brokers. Note this must *not* include the **@QPID** suffix.
ha-password *PASS*	Password for HA brokers.
ha-mechanism *MECHANISM*	Mechanism for HA brokers. Any mechanism you enable for broker-to-broker communication can also be used by a client, so do not use **ANONYMOUS** in a secure environment.

This identity is used to authorize federation links from backup to primary. It is also used to authorize actions on the backup to replicate primary state, for example creating queues and exchanges.

When authorization is enabled you must have an Access Control List with the following rule to allow HA replication to function. Suppose **ha-username**=*USER*

```
acl allow USER@QPID all all
```

See Also:

- Section 9.1.8, “ACL Requirements for Clustering”

Report a bug

9.1.20. HA Clustering and Persistence

If you use a persistent store for your messages then each broker in a cluster will have its own store. If the entire cluster fails and is restarted, the *first* broker that becomes primary will recover from its store. All the other brokers will clear their stores and get an update from the primary to ensure consistency.

Report a bug

9.1.21. Queue Replication and HA

As well as support for an active-passive cluster, the **HA** module supports individual queue replication, even if the brokers are not in a clustered environment. The original queue is used as normal, however the replica queue is updated automatically as messages are added to or removed from the original queue.

To create a replica queue, the **HA** module must be loaded on both the original and replica brokers, which is done automatically by default.

For standalone brokers, the **ha-queue-replication=yes** configuration option must be specified. This option is not required for brokers that are part of a clustered environment, because the option is loaded automatically.

Important

Queue modification must only be received through automatic updates from the original queue.

Manually adding or removing messages on the replica queue will make replication inconsistent, and may cause message loss.

The **HA** module does not enforce restricted access to the replica queue (as it does in the case of a cluster). The application must ensure the replica is not used until it has been disconnected from the original.

Example 9.1. Replicate a Queue Between Nodes

Suppose that **myqueue** is a queue on **node1**.

To create a replica of **myqueue** on **node2**, run the following command:

```
qpid-config --broker=node2 add queue --start-replica node1 myqueue
```

If **myqueue** already exists on the replica broker, run the following command to start replication from the original queue:

```
qpid-ha replicate -b node2 node1 myqueue
```

Report a bug

9.2. Cluster management

9.2.1. Cluster Management using qpid-ha

qpid-ha is a command-line utility that allows you to view information on a cluster and its brokers, disconnect a client connection, shut down a broker in a cluster, or shut down the entire cluster. It accepts a command and options.

qpid-ha has the following commands and parameters:

Commands

status

Print HA status. Returns information whether the specified broker is acting as a primary (active) or a backup (ready). With the **`--all`** option will list the whole cluster.

Examples:

```
# qpid-ha status
ready
# qpid-ha status --all
192.168.6.60:5672 ready
192.168.6.61:5672 active
192.168.6.62:5672 ready
```

ping

Check if the broker is alive and responding.

query

Print HA configuration and status. The following information is returned:

- broker status: primary (active) or backup (ready).
- list of HA broker URLs
- public (virtual) HA URL
- replication status

Example:

```
# qpid-ha query
Status:        ready
Brokers URL:
amqp:tcp:192.168.6.60:5672,tcp:192.168.6.61:5672,tcp:192.168.6.62:567
2
Public URL:    amqp:tcp:192.168.6.251:5672
Replicate:     all
```

replicate

Set up replication from <*queue*> on <*remote-broker*> to <*queue*> on the current broker.

Parameters

--broker=*BROKER*

The address of qpidd broker. The syntax is shown below:

```
[username/password@] hostname | ip-address [:port]
```

--sasl-mechanism=*SASL_MECH*

SASL mechanism for authentication (e.g. EXTERNAL, ANONYMOUS, PLAIN, CRAM-MD5, DIGEST-MD5, GSSAPI). SASL automatically picks the most secure available mechanism - use this option to override.

--ssl-certificate=*SSL_CERT*

Client SSL certificate (PEM Format)

Client SSL certificate (PEM Format).

--config=*CONFIG*

Connect to the local qpidd by reading its configuration file (**`/etc/qpid/qpidd.conf`**, for example) .

--timeout=*SECONDS*

Give up if the broker does not respond within the timeout. **`0`** means wait forever. The default is **`10.0`**.

--ssl-key=*KEY*

Client SSL private key (PEM Format)

--help-all

Outputs all of the above commands and parameters.

Each command accepts a variety of options. You can see the options using the **`--help-all`** option:

```
$ qpid-ha --help-all
```

See Also:

- Section 9.1.21, “Queue Replication and HA”

Report a bug

9.3. Cluster Troubleshooting

9.3.1. Troubleshooting Cluster configuration

When you initially start a HA cluster, all brokers are in joining mode. The brokers do not automatically select a primary, they rely on the cluster manager **rgmanager** to do so. If **rgmanager** is not running or is not configured correctly, brokers will remain in the joining state. Refer to Section 9.1.9, “Cluster Resource Manager (rgmanager)” for more information.

If a broker is unable to establish a connection to another broker in the cluster, the log will contain SASL errors similar to the following:

```
info SASL: Authentication failed: SASL(-13): user not found: Password
verification failed
```

```
warning Client closed connection with 320: User anonymous@QPID federation
connection denied. Systems with authentication enabled must specify ACL
create link rules.
```

```
warning Client closed connection with 320: ACL denied anonymous@QPID creating
a federation link.
```

Procedure 9.3. Troubleshooting Cluster SASL Configuration

1. Set the HA security configuration and ACL file as described in Section 9.1.19, “Security”

2. Once the cluster is running and the primary is promoted , run:

```
qpid-ha status --all
```

   to ensure that the brokers are running as one cluster.

3. Once the cluster is running, run **qpid-ha** to make sure that the brokers are running as one cluster.

Report a bug

9.3.2. Slow Recovery Times

The following configuration settings affect recovery time. The example values used give fast recovery on a lightly loaded system. Run tests to determine if the appropriate values for your system and load conditions.

cluster.conf

<rm status_poll_interval=1>

status_poll_interval is the interval in seconds that the resource manager checks the status of managed services. This affects how quickly the manager will detect failed services.

<ip address="20.0.20.200" monitor_link="yes" sleeptime="0"/>

This is a virtual IP address for client traffic. **monitor_link="yes"** means monitor the health of the NIC used for the VIP. **sleeptime="0"** means don't delay when failing over the VIP to a new address.

qpidd.conf

link-maintenance-interval=0.1

The number of seconds to wait for back-up brokers to check the link to the primary re-connect if required. This value defaults to **2**. The value can be set lower for a faster failover (for example, **0.1**).

Note

Setting the value too low will result in excessive link-checking activity on the brokers.

link-heartbeat-interval=5

Heartbeat interval for federation links. The HA cluster uses federation links between the primary and each backup. The primary can take up to twice the heartbeat interval to detect a failed backup. When a sender sends a message the primary waits for all backups to acknowledge before acknowledging to the sender. A disconnected backup may cause the primary to block senders until it is detected via heartbeat.

This interval is also used as the timeout for broker status checks by rgmanager. It may take up to this interval for **rgmanager** to detect a hung broker.

The default is 120 seconds. This may be too high for many productions scenarios where availability and response time is important. However, if set too low, under network congestion or heavy load a slow-to-respond broker may be restarted by **rgmanager**.

Report a bug

9.3.3. Total Cluster Failure

The cluster guarantees availability as long as one active primary broker or ready backup broker is left alive. If all brokers fail simultaneously, the cluster fails and non-persistent data is lost.

Brokers are in one of 6 states:

1. **`standalone`**: not part of a HA cluster
2. **`joining`**: newly started backup, not yet joined to the cluster.
3. **`catch-up`**: backup has connected to the primary and is downloading queues, messages etc.
4. **`ready`**: backup is connected and actively replicating from primary, it is ready to take over.
5. **`recovering`**: newly-promoted to primary, waiting for backups to catch up before serving clients. Only a single primary broker can be recovering at a time.
6. **`active`**: serving clients, only a single primary broker can be active at a time.

While there is an active primary broker, clients can get service. If the active primary fails, one of the "ready" backup brokers takes over, recovers and becomes active. A backup can only be promoted to primary if it is in the "ready" state (with the exception of the first primary in a new cluster where all brokers are in the "joining" state)

Given a stable cluster of N brokers with one active primary and N-1 ready backups, the system can sustain N-1 failures in rapid succession. The surviving broker will be promoted to active and continue to give service.

However, at this point the system cannot sustain a failure of the surviving broker until at least one of the other brokers recovers, catches up and becomes a ready backup. If the surviving broker fails before that the cluster will fail in one of two modes (depending on the exact timing of failures).

1. The cluster hangs

All brokers are in joining or catch-up mode. **`rgmanager`** tries to promote a new primary but cannot find any candidates and so gives up. clustat will show that the **`qpidd`** services are running but the the **`qpidd-primary`** service has stopped, something like this:

Table 9.3.

Service Name	Owner (Last)	State
service:mrg33-qpidd-service	20.0.10.33	started
service:mrg34-qpidd-service	20.0.10.34	started
service:mrg35-qpidd-service	20.0.10.35	started
service:qpidd-primary-service	(20.0.10.33)	stopped

Eventually all brokers become stuck in "joining" mode, as shown by **`qpid-ha status --all`**.

At this point you need to restart the cluster in one of the following ways:

Restart the entire cluster

- In luci:<your-cluster>:Nodes click reboot to restart the entire cluster.
- *or* stop and restart the cluster with **`ccs --stopall`**; **`ccs --startall`**

Restart just the Qpid services

- In luci:<your-cluster>:Service Groups:
  - select all the qpidd (not primary) services, click restart.
  - select the qpidd-primary service, click restart.
- *or* stop the primary and qpidd services with clusvcadm, then restart (primary last)

2. The cluster reboots

A new primary is promoted and the cluster is functional. All non-persistent data from before the failure is lost.

Report a bug

9.3.4. Fencing and Network Partitions

A network partition is a a network failure that divides the cluster into two or more sub-clusters, where each broker can communicate with brokers in its own sub-cluster but not with brokers in other sub-clusters. This condition is also referred to as a "split brain".

Nodes in one sub-cluster can't tell whether nodes in other sub-clusters are dead or are still running but disconnected. We cannot allow each sub-cluster to independently declare its own qpidd primary and start serving clients, as the cluster will become inconsistent. We must ensure only one sub-cluster continues to provide service.

A *quorum* determines which sub-cluster continues to operate, and*power fencing* ensures that nodes in non-quorate sub-clusters cannot attempt to provide service inconsistently. For more information refer to the *Red Hat Enterprise Linux 6 High Availability Add-on Overview* Chapter 2. Quorum and Chapter 4. Fencing.

Report a bug

Chapter 10. Broker Federation

10.1. Broker Federation

Broker Federation allows messaging networks to be defined by creating *message routes*, in which messages in one broker (the *source broker*) are automatically routed to another broker (the *destination broker*). These routes can be defined between exchanges in the two brokers (the *source exchange* and the *destination exchange*), or from a queue in the source broker (the *source queue*) to an exchange in the destination broker.

Message routes are unidirectional; when bidirectional flow is needed, one route is created in each direction. Routes can be durable or transient. A durable route survives broker restarts, restoring a route as soon as both the source broker and the destination are available. If the connection to a destination is lost, messages associated with a durable route continue to accumulate on the source, so they can be retrieved when the connection is reestablished.

Broker Federation can be used to build large messaging networks, with many brokers, one route at a time. If network connectivity permits, an entire distributed messaging network can be configured from a single location. The rules used for routing can be changed dynamically as servers or responsibilities change at different times of day, or to reflect other changing conditions.

Report a bug

10.2. Broker Federation Use Cases

Broker Federation is useful in a wide variety of scenarios. Some of these have to do with functional organization; for instance, brokers can be organized by geography, service type, or priority. Here are some use cases for federation:

- Geography: Customer requests can be routed to a processing location close to the customer.
- Service Type: High value customers can be routed to more responsive servers.
- Load balancing: Routing among brokers can be changed dynamically to account for changes in actual or anticipated load.
- High Availability: Routing can be changed to a new broker if an existing broker becomes unavailable.
- WAN Connectivity: Federated routes can connect disparate locations across a wide area network, while clients connect to brokers on their own local area network. Each broker can provide persistent queues that can hold messages even if there are gaps in WAN connectivity.
- Functional Organization: The flow of messages among software subsystems can be configured to mirror the logical structure of a distributed application.
- Replicated Exchanges: High-function exchanges like the XML exchange can be replicated to scale performance.
- Interdepartmental Workflow: The flow of messages among brokers can be configured to mirror interdepartmental workflow at an organization.

Report a bug

10.3. Broker Federation Overview

10.3.1. Message Routes

Broker Federation is done by creating message routes. The destination for a route is always an exchange on the destination broker.

By default, a message route is created by configuring the destination broker, which then contacts the source broker to subscribe to the source queue. This is called a *pull route*.

It is also possible to create a route by configuring the source broker, which then contacts the destination broker to send messages. This is called a *push route*, and is particularly useful when the destination broker may not be available at the time the messaging route is configured, or when a large number of routes are created with the same destination exchange.

The source for a route can be either an exchange or a queue on the source broker. If a route is between two exchanges, the routing criteria can be given explicitly, or the bindings of the destination exchange can be used to determine the routing criteria. To support this functionality, there are three kinds of message routes:

- Queue routes
- Exchange routes
- Dynamic exchange routes

Report a bug

10.3.2. Queue Routes

Queue Routes route all messages from a source queue to a destination exchange. If message acknowledgment is enabled, messages are removed from the queue when they have been received by the destination exchange; if message acknowledgment is off, messages are removed from the queue when sent.

When there are multiple subscriptions on an AMQP queue, messages are distributed among subscribers. For example, two queue routes being fed from the same queue will each receive a load-balanced number of messages.

If fanout behavior is required instead of load-balancing, use an *exchange route*.

Report a bug

10.3.3. Exchange Routes

Exchange routes route messages from a source exchange to a destination exchange, using a binding key (which is optional for a fanout exchange).

Internally, creating an exchange route creates a private queue (auto-delete, exclusive) on the source broker to hold messages that are to be routed to the destination broker, binds this private queue to the source broker exchange, and subscribes the destination broker to the queue.

Report a bug

10.3.4. Dynamic Exchange Routes

A *dynamic exchange route* connects an exchange on one broker (the source) to an exchange on another broker (the destination) so that a client can subscribe to the exchange on the destination broker, and receive messages that match that subscription sent to either exchange.

Dynamic exchange routes are *directional*. When a dynamic exchange route is created from broker A to broker B, broker B dynamically creates and deletes subscriptions to broker A's exchange on behalf of its

clients. In this way broker B acts as a proxy for its clients. When a client subscribes to the destination exchange on broker B, broker B subscribes the exchange to broker A's source exchange with the subscription that the client created. Clients subscribing to broker B effectively subscribe to both the exchange on broker B and the dynamically routed exchange on broker A.

The consuming broker creates and deletes subscriptions to the source broker's dynamically routed exchange as clients subscribe and unsubscribe to the destination exchange on the consuming broker. It is important to note that the dynamic exchange route is a competing subscription, so the destination broker is a message consumer like any other subscriber.

In a dynamic exchange route, the source and destination exchanges must have the same exchange type, and they must have the same name. For instance, if the source exchange is a direct exchange, the destination exchange must also be a direct exchange, and the names must match.

Internally, dynamic exchange routes are implemented in the same way as exchange routes, except that the bindings used to implement dynamic exchange routes are modified if the bindings in the destination exchange change.

A dynamic exchange route is always a pull route. It can never be a push route.

See Also:

- Section 10.4.7, “Create and Delete Dynamic Exchange Routes”

Report a bug

10.3.5. Federation Topologies

A federated network is generally a tree, star, or line, using bidirectional links (implemented as a pair of unidirectional links) between any two brokers. A ring topology is also possible, if only unidirectional links are used.

Every message transfer takes time. For better performance, minimize the number of brokers between the message origin and final destination. In most cases, tree or star topologies do this best.

For any pair of nodes A, B in a federated network, if there is more than one path between A and B, ensure that it is not possible for any messages to cycle between A and B. Looping message traffic can flood the federated network. The tree, star and line topologies do not have message loops. A ring topology with bidirectional links is one example of a topology that causes this problem, because a given broker can receive the same message from two different brokers. Mesh topologies can also cause this problem.

Report a bug

10.3.6. Federation Among High Availability Clusters

Federation is generally used together with High Availability Message Clusters, using clusters to provide high availability on each LAN, and federation to route messages among the clusters.

To create a message route between two clusters, create a route between any one broker in the first cluster and any one broker in the second cluster. Each broker in a given cluster can use message routes defined for another broker in the same cluster. If the broker for which a message route is defined should fail, another broker in the same cluster can restore the message route.

Report a bug

10.4. Configuring Broker Federation

10.4.1. The qpid-route Utility

qpid-route is a command line utility used to configure federated networks of brokers and to view the status and topology of networks. It can be used to configure routes among any brokers that **qpid-route** can connect to.

Report a bug

10.4.2. qpid-route Syntax

The syntax of **qpid-route** is as follows:

```
qpid-route [OPTIONS] dynamic add <dest-broker> <src-broker> <exchange>
qpid-route [OPTIONS] dynamic del <dest-broker> <src-broker> <exchange>

qpid-route [OPTIONS] route add <dest-broker> <src-broker> <exchange>
<routing-key>
qpid-route [OPTIONS] route del <dest-broker> <src-broker> <exchange>
<routing-key>

qpid-route [OPTIONS] queue add <dest-broker> <src-broker> <dest-exchange>
<src-queue>
qpid-route [OPTIONS] queue del <dest-broker> <src-broker> <dest-exchange>
<src-queue>

qpid-route [OPTIONS] route list  [<broker>]
qpid-route [OPTIONS] route flush [<broker>]
qpid-route [OPTIONS] route map   [<broker>]

qpid-route [OPTIONS] link add  <dest-broker> <src-broker>
qpid-route [OPTIONS] link del  <dest-broker> <src-broker>
qpid-route [OPTIONS] link list [<dest-broker>]
```

The syntax for **broker**, **dest-broker**, and **src-broker** is as follows:

```
[username/password@] hostname | ip-address [:<port>]
```

The following are all valid examples of the above syntax: **localhost**, **10.1.1.7:10000**, **broker-host:10000**, **guest/guest@localhost**.

Report a bug

10.4.3. qpid-route Options

Changes

- New for MRG-M 3.

Table 10.1. qpid-route options to manage federation

Option	Description
-v	Verbose output.
-q	Quiet output, will not print duplicate warnings.
-d	Make the route durable.

Option	Description
-e	Delete link after deleting the last route on the link.
--timeout *N*	Maximum time to wait when qpid-route connects to a broker, in seconds. Default is 10 seconds.
--ack *N*	Acknowledge transfers of routed messages in batches of N. Default is 0 (no acknowledgments). Setting to 1 or greater enables acknowledgments; when using acknowledgments, values of N greater than 1 can significantly improve performance, especially if there is significant network latency between the two brokers.
--credit *N*	Specifies a finite credit to use with acknowledgements. By default, credit is 0 and credit flow control is disabled. Backpressure can be tuned by means of an explicit **--credit** argument.
-s [--src-local]	Configure the route in the source broker (create a push route).
-t <transport> [--transport <transport>]	Transport protocol to be used for the route. - tcp (default) - ssl - rdma
--client-sasl-mechanism <mech>	SASL mechanism for authentication when the client connects to the destination broker (for example: EXTERNAL, ANONYMOUS, PLAIN, CRAM-MD, DIGEST-MD5, GSSAPI).

Report a bug

10.4.4. Create and Delete Queue Routes

1. To create and delete queue routes, use the following syntax:

```
qpid-route [OPTIONS] queue add <dest-broker> <src-broker> <dest-
exchange> <src-queue>
qpid-route [OPTIONS] queue del <dest-broker> <src-broker> <dest-
exchange> <src-queue>
```

2. For example, use the following command to create a queue route that routes all messages from the queue named **public** on the source broker **localhost:10002** to the **amq.fanout** exchange on the destination broker **localhost:10001**:

```
$ qpid-route queue add localhost:10001 localhost:10002 amq.fanout
public
```

3. Optionally, specify the **-d** option to persist the queue route. The queue route will be restored if one or both of the brokers is restarted:

```
$ qpid-route -d queue add localhost:10001 localhost:10002 amq.fanout
public
```

4. The **del** command takes the same arguments as the **add** command. Use the following command to delete the queue route described above:

```
$ qpid-route queue del localhost:10001 localhost:10002 amq.fanout
public
```

Report a bug

10.4.5. Create and Delete Exchange Routes

1. To create and delete exchange routes, use the following syntax:

```
qpid-route [OPTIONS] route add <dest-broker> <src-broker> <exchange>
<routing-key>
qpid-route [OPTIONS] route del <dest-broker> <src-broker> <exchange>
<routing-key>
qpid-route [OPTIONS] flush [<broker>]
```

2. For example, use the following command to create an exchange route that routes messages that match the binding key **global.#** from the **amq.topic** exchange on the source broker **localhost:10002** to the **amq.topic** exchange on the destination broker **localhost:10001**:

```
$ qpid-route route add localhost:10001 localhost:10002 amq.topic
global.#
```

3. In many applications, messages published to the destination exchange must also be routed to the source exchange. Create a second exchange route, reversing the roles of the two exchanges:

```
$ qpid-route route add localhost:10002 localhost:10001 amq.topic
global.#
```

4. Specify the **-d** option to persist the exchange route. The exchange route will be restored if one or both of the brokers is restarted:

```
$ qpid-route -d route add localhost:10001 localhost:10002 amq.fanout
public
```

5. The **del** command takes the same arguments as the **add** command. Use the following command to delete the first exchange route described above:

```
$ qpid-route route del localhost:10001 localhost:10002 amq.topic
global.#
```

Report a bug

10.4.6. Delete All Routes for a Broker

- Use the **flush** command to delete all routes for a given broker:

```
qpid-route [OPTIONS] route flush [<broker>]
```

For example, use the following command to delete all routes for the broker **localhost:10001**:

```
$ qpid-route route flush localhost:10001
```

Report a bug

10.4.7. Create and Delete Dynamic Exchange Routes

1. To create and delete dynamic exchange routes, use the following syntax:

   ```
   qpid-route [OPTIONS] dynamic add <dest-broker> <src-broker> <exchange>
   qpid-route [OPTIONS] dynamic del <dest-broker> <src-broker> <exchange>
   ```

2. Create a new topic exchange on each of two brokers:

   ```
   $ qpid-config -a localhost:10003 add exchange topic fed.topic
   $ qpid-config -a localhost:10004 add exchange topic fed.topic
   ```

3. Create a dynamic exchange route that routes messages from the **`fed.topic`** exchange on the source broker **`localhost:10004`** to the **`fed.topic`** exchange on the destination broker **`localhost:10003`**:

   ```
   $ qpid-route dynamic add localhost:10003 localhost:10004 fed.topic
   ```

   Internally, this creates a private autodelete queue on the source broker, and binds that queue to the **`fed.topic`** exchange on the source broker, using each binding associated with the **`fed.topic`** exchange on the destination broker.

4. In many applications, messages published to the destination exchange must also be routed to the source exchange. Create a second dynamic exchange route, reversing the roles of the two exchanges:

   ```
   $ qpid-route dynamic add localhost:10004 localhost:10003 fed.topic
   ```

5. Specify the **`-d`** option to persist the exchange route. The exchange route will be restored if one or both of the brokers is restarted:

   ```
   $ qpid-route -d dynamic add localhost:10004 localhost:10003 fed.topic
   ```

   When an exchange route is durable, the private queue used to store messages for the route on the source exchange is also durable. If the connection between the brokers is lost, messages for the destination exchange continue to accumulate until it can be restored.

6. The **`del`** command takes the same arguments as the **`add`** command. Delete the first exchange route described above:

   ```
   $ qpid-route dynamic del localhost:10004 localhost:10003 fed.topic
   ```

   Internally, this deletes the bindings on the source exchange for the private queues associated with the message route.

Report a bug

10.4.8. View Routes

Procedure 10.1. Using the `route list` command

1. Create the following two routes:

```
$ qpid-route dynamic add localhost:10003 localhost:10004 fed.topic
$ qpid-route dynamic add localhost:10004 localhost:10003 fed.topic
```

2. Use the **`route list`** command to show the routes associated with the broker:

```
$ qpid-route route list localhost:10003
localhost:10003 localhost:10004 fed.topic <dynamic>
```

   Note that this shows only one of the two routes created, namely the route for which **`localhost:10003`** is a destination.

3. To view the route for which **`localhost:10004`** is a destination, run **`route list`** on **`localhost:10004`**:

```
$ qpid-route route list localhost:10004
localhost:10004 localhost:10003 fed.topic <dynamic>
```

Procedure 10.2. Using the `route map` command

1. The **`route map`** command shows all routes associated with a broker, and recursively displays all routes for brokers involved in federation relationships with the given broker. For example, run the **`route map`** command for the two brokers configured above:

```
$ qpid-route route map localhost:10003

Finding Linked Brokers:
    localhost:10003... Ok
    localhost:10004... Ok

Dynamic Routes:

  Exchange fed.topic:
    localhost:10004 <=> localhost:10003

Static Routes:
  none found
```

   Note that the two dynamic exchange links are displayed as though they were one bidirectional link. The **`route map`** command is helpful for larger, more complex networks.

2. Configure a network with 16 dynamic exchange routes:

```
qpid-route dynamic add localhost:10001 localhost:10002 fed.topic
qpid-route dynamic add localhost:10002 localhost:10001 fed.topic

qpid-route dynamic add localhost:10003 localhost:10002 fed.topic
qpid-route dynamic add localhost:10002 localhost:10003 fed.topic

qpid-route dynamic add localhost:10004 localhost:10002 fed.topic
qpid-route dynamic add localhost:10002 localhost:10004 fed.topic
```

```
qpid-route dynamic add localhost:10002 localhost:10005 fed.topic
qpid-route dynamic add localhost:10005 localhost:10002 fed.topic

qpid-route dynamic add localhost:10005 localhost:10006 fed.topic
qpid-route dynamic add localhost:10006 localhost:10005 fed.topic

qpid-route dynamic add localhost:10006 localhost:10007 fed.topic
qpid-route dynamic add localhost:10007 localhost:10006 fed.topic

qpid-route dynamic add localhost:10006 localhost:10008 fed.topic
qpid-route dynamic add localhost:10008 localhost:10006 fed.topic
```

3. Use the **`route map`** command starting with any one broker to see the entire network:

```
$ qpid-route route map localhost:10001

Finding Linked Brokers:
    localhost:10001... Ok
    localhost:10002... Ok
    localhost:10003... Ok
    localhost:10004... Ok
    localhost:10005... Ok
    localhost:10006... Ok
    localhost:10007... Ok
    localhost:10008... Ok

Dynamic Routes:

  Exchange fed.topic:
    localhost:10002 <=> localhost:10001
    localhost:10003 <=> localhost:10002
    localhost:10004 <=> localhost:10002
    localhost:10005 <=> localhost:10002
    localhost:10006 <=> localhost:10005
    localhost:10007 <=> localhost:10006
    localhost:10008 <=> localhost:10006

Static Routes:
  none found
```

Report a bug

10.4.9. Resilient Connections

When a broker route is created, or when a durable broker route is restored after broker restart, a *resilient connections* is created between the source broker and the destination broker. If the connection fails due to a communication error, it attempts to reconnect. The retry interval begins at 2 seconds and, as more attempts are made, grows to 64 seconds. It continues to retry every 64 seconds thereafter. If the connection fails due to an authentication problem, it will not attempt to reconnect.

Report a bug

10.4.10. View Resilient Connections

The command **link list** can be used to show the resilient connections for a broker:

```
$ qpid-route link list localhost:10001

Host            Port    Transport Durable  State             Last Error
=============================================================================
=
localhost       10002   tcp          N     Operational
localhost       10003   tcp          N     Operational
localhost       10009   tcp          N     Waiting           Connection
refused
```

In the above output, **Last Error** contains the string representation of the last connection error received for the connection. **State** represents the state of the connection, and may be one of the following values:

Table 10.2. State values in $ qpid-route link list

Option	Description
Waiting	Waiting before attempting to reconnect.
Connecting	Attempting to establish the connection.
Operational	The connection has been established and can be used.
Failed	The connection failed and will not retry (usually because authentication failed).
Closed	The connection has been closed and will soon be deleted.
Passive	If a cluster is federated to another cluster, only one of the nodes has an actual connection to remote node. Other nodes in the cluster have a passive connection.

Report a bug

10.4.11. Broker Federation Limitations Between 2.x and 3.x

Version 2.x to version 3.x broker federation support continues to be provided by the long-standing 0-10 federation capability. To administer both 2.x and 3.x brokers for federation, a version 3.x **qpid-route** must be provisioned to ensure backwards compatibility with 2.x brokers. This is due to differences in argument count mismatch handling of 2.x brokers.

If an older version of **qpid-route** attempts to create a link where the source broker uses a newer version, compatibility will break because 2.x versions of qpid-route do not include the improved argument count handling.

Report a bug

Chapter 11. Qpid JCA Adapter

11.1. JCA Adapter

A JCA Adapter provides outbound and inbound connectivity between Enterprise Information Systems (for example, mainframe transaction processing and database systems), application servers, and enterprise applications. It controls the inflow of messages to Message-Driven Beans (MDBs) and the outflow of messages sent from other Java EE components. It also provides a variety of options to fine tune your messaging applications.

Report a bug

11.2. Qpid JCA Adapter

The Qpid Resource Adapter is a Java Connector Architecture (JCA) 1.5 compliant resource adapter that permits Java EE integration between EE applications and AMQP 0.10 message brokers. Currently the adapter only supports C++ based brokers and has only been tested with Apache Qpid C++ broker.

Report a bug

11.3. Install the Qpid JCA Adapter

For systems running Red Hat Enterprise Linux 5 or 6, the Qpid JCA Adapter is provided in the **`qpid-jca`** and **`qpid-jca-xarecovery`** packages. These RPM packages are included with the default MRG Messaging installation.

For other operating systems, the Qpid JCA Adapter is provided in the **`JCA Adapter <JCA-VERSION>`** and **`JCA Adapter <JCA-VERSION> detached signature`** packages. These ZIP files can be obtained from the **`Downloads`** section of the **`MRG Messaging v. 2 (for non-Linux platforms)`** channel on the Red Hat Network.

Report a bug

11.4. Qpid JCA Adapter Configuration

11.4.1. Per-Application Server Configuration Information

Deploying the components of a resource adapter varies with different application servers. This guide provides an overview of the adapter's capabilities, and installation instructions for JBoss Enterprise Application Platform 5 and 6. For most application server platforms, platform-specific details for configuring the adapter are provided in README files typically named **`README-<server-platform>.txt`**.

See Also:

- Section 11.5, “Deploying the Qpid JCA Adapter on JBoss EAP 5”
- Section 11.6, “Deploying the Qpid JCA Adapter on JBoss EAP 6”

Report a bug

11.4.2. JCA Adapter ra.xml Configuration

The **ra.xml** file contains configuration parameters for the JCA Adapter. In some application server environments this file is edited directly to change configuration, in other environments (such as JBoss EAP 5) it is overridden by configuration in a ***-ds.xml** file.

The following are the properties set in the **ra.xml** file that can be configured or overridden:

Table 11.1. ResourceAdapter Properties

Parameter	Description	Default Value
ClientId	Client ID for the connection	client_id
SetupAttempts	Number of setup attempts before failing	5
SetupInterval	Interval between setup attempts in milliseconds	5000
UseLocalTx	Use local transations rather than XA	false
Host	Broker host	localhost
Port	Broker port	5672
Path	Virtual Path for Connection Factory	test
ConnectionURL	Connection URL	amqp://anonymous:passwd@client/test?brokerlist='tcp://localhost?sasl_mechs='PLAIN''
UseConnectionPerHandler	Use a JMS Connection per MessageHandler	true

Table 11.2. Outbound ResourceAdapter Properties

Parameter	Description	Default Value
SessionDefaultType	Default session type	javax.jms.Queue
UseTryLock	Specify lock timeout in seconds	0
UseLocalTx	Use local transactions rather than XA	false
ClientId	Client ID for the connection	client_id
ConnectionURL	Connection URL	
Host	Broker host	localhost
Port	Broker port	5672
Path	Virtual Path for Connection Factory	test

Inbound ResourceAdapter and AdminObjects

Additionally, the **ra.xml** file contains configuration for the Inbound Resource Adapter and Administered Objects (**AdminObject**). The configuration for these two differs from the Resource Adapter and Outbound Resource Adapter configuration. The Resource Adapter and Outbound Resource Adapter configuration sets values for configuration parameters in the **ra.xml** file. The Inbound Resource Adapter and Admin Object *define* configuration parameters, but do not set them. It is the responsibility of the Administered Object mbean definition to set the properties defined in **ra.xml**.

Table 11.3. AdminObject Properties

AdminObject Class	Property
`org.apache.qpid.ra.admin.QpidQueue`	`DestinationAddress`
`org.apache.qpid.ra.admin.QpidTopic`	`DestinationAddress`
`org.apache.qpid.ra.admin.QpidConnectionFactoryProxy`	`ConnectionURL`

See Also:

- Section 11.5, “Deploying the Qpid JCA Adapter on JBoss EAP 5”
- Section 11.6, “Deploying the Qpid JCA Adapter on JBoss EAP 6”

Report a bug

11.4.3. Transaction Support

The Qpid JCA Adapter provides three levels of transaction support — **`XA`**, **`LocalTransactions`** and **`NoTransaction`**.

Typically, using the Qpid JCA Adapter implies the use of XA transactions, however this is not always the case. Transaction support configuration is specific to each application server. Refer to the documentation for each supported application server.

Report a bug

11.4.4. Transaction Limitations

These are the limitations with the Qpid JCA Adapter:

1. The Qpid C++ broker does not support the use of XA within the context of clustered brokers. If you are running a cluster, you must configure the adapter to use LocalTransactions.
2. XARecovery is currently not implemented. In case of a system failure, incomplete (or *in doubt*) transactions must be manually resolved by an administrator or other qualified personnel.

Report a bug

11.5. Deploying the Qpid JCA Adapter on JBoss EAP 5

11.5.1. Deploy the Qpid JCA adapter on JBoss EAP 5

To install the Qpid JCA Adapter for use with JBoss EAP 5, transfer its configuration files to the JBoss deploy directory.

Procedure 11.1. To deploy the Qpid JCA adapter for JBoss EAP

1. Locate the **`qpid-ra-<version>.rar`** file. It is a zip archive data file which contains the resource adapter, Qpid Java client **`.jar`** files and the **`META-INF`** directory.
2. Copy the **`qpid-ra-<version>.rar`** file to your JBoss deploy directory. The JBoss deploy directory is ***`JBOSS_ROOT`*`/server/<server-name>/deploy`**, where *JBOSS_ROOT* denotes the root directory of your JBoss installation and <*server-name*> denotes the name of your deployment server.
3. A successful adapter installation is accompanied by the following message:

```
INFO  [QpidResourceAdapter] Qpid resource adaptor started
```

At this point, the adapter is deployed and ready for configuration.

Report a bug

11.5.2. JCA Configuration on JBoss EAP 5

11.5.2.1. JCA Adapter Configuration File

Changes

- Updated April 2013.

The standard configuration mechanism for JCA adapters in the EAP 5.x environment is the **`*-ds.xml`** file. The Qpid JCA adapter has a global **`ra.xml`** file, per the JCA specification, but the default set of values in this file are almost always overridden via the **`*-ds.xml`** file configuration file.

The **`ResourceAdapter`** configuration provides generic properties for inbound and outbound connectivity. However, these properties can be overridden when deploying **`ManagedConnectionFactories`** and inbound activations using the standard JBoss configuration artifacts, the **`*-ds.xml`** file and MDB ActivationSpec. A sample **`*-ds.xml`** file, **`qpid-jca-ds.xml`**, is located in the directory, **`/usr/share/doc/qpid-jca-<VERSION>/example/conf/`**.

The Qpid JCA Adapter directory **`/usr/share/qpid-jca`** contains the general **`README.txt`** file, which provides a detailed description of all the properties associated with the Qpid JCA Adapter.

Report a bug

11.5.2.2. ConnectionFactory Configuration

11.5.2.2.1. ConnectionFactory

Compliant with the JCA specification, the ConnectionFactory component defines properties for standard outbound connectivity.

Report a bug

11.5.2.2.2. ConnectionFactory Configuration in EAP 5

In JBoss EAP 5, ConnectionFactories are configured using the **`*-ds.xml`** file. A sample file (**`qpid-jca-ds.xml`**) is provided with your distribution. This file can be modified to suit your development or deployment needs.

Report a bug

11.5.2.2.3. XAConnectionFactory Example

The following example describes the ConnectionFactory portion of the sample file for XA transactions:

```
<tx-connection-factory>
  <jndi-name>QpidJMSXA</jndi-name>
  <xa-transaction/>
  <rar-name>qpid-ra-<ra-version>.rar</rar-name>
  <connection-
definition>org.apache.qpid.ra.QpidRAConnectionFactory</connection-
```

```
definition>
  <config-property name="ConnectionURL">amqp://guest:guest@/test?
brokerlist='tcp://localhost:5672?sasl_mechs='ANONYMOUS''</config-property>
  <max-pool-size>20</max-pool-size>
</tx-connection-factory>
```

- The ***QpidJMSXA*** connection factory defines an XA-capable ManagedConnectionFactory.
- You must insert your particular ra version for the ***rar-name*** property.
- The ***jndi-name*** and ***ConnectionURL*** properties can be modified to suit your environment.

After deployment, the ConnectionFactory will be bound into Java Naming and Directory Interface (JNDI) with the following syntax:

```
java:<jndi-name>
```

In this example, this would resolve to:

```
java:QpidJMSXA
```

Report a bug

11.5.2.2.4. Local ConnectionFactory Example

The following example describes the **ConnectionFactory** portion of the sample file for local transactions:

```
<tx-connection-factory>
   <jndi-name>QpidJMS</jndi-name>
   <rar-name>
     qpid-ra-<ra-version>.rar
   </rar-name>
   <local-transaction/>
   <config-property name="useLocalTx" type="java.lang.Boolean">
      true
   </config-property>
   <config-property name="ConnectionURL">
amqp://anonymous:@client/test?brokerlist='tcp://localhost:5672?
sasl_mechs='ANONYMOUS''
   </config-property>
   <connection-definition>
      org.apache.qpid.ra.QpidRAConnectionFactory
   </connection-definition>
   <max-pool-size>20<max-pool-size>
</tx-connection-factory>
```

- The ***QpidJMS*** connection factory defines a non-XA **ConnectionFactory**. Typically this is used as a specialized **ConnectionFactory** where XA is not desired, or if you are running with a clustered Qpid Broker configuration that currently does not support XA.
- You must insert your particular ra version for the ***rar-name*** property.
- The ***jndi-name*** and ***ConnectionURL*** properties can be modified to suit your environment.

After deployment, the **ConnectionFactory** will be bound into Java Naming and Directory Interface (JNDI) with the following syntax:

```
java:<jndi-name>
```

In this example, this would resolve to:

```
java:QpidJMS
```

Report a bug

11.5.2.3. Administered Object Configuration

11.5.2.3.1. Administered Objects in EAP 5

Destinations (queues, topics) are configured in JBoss EAP via JCA standard Administered Objects (AdminObjects). These objects are placed within the **`*-ds.xml`** file alongside your ConnectionFactory configurations. The sample **`qpid-jca-ds.xml`** file provides two such objects: JMS Queue/ Topic and Connection Factory.

Report a bug

11.5.2.3.2. JMS Queue Administered Object Example

Changes

- Updated April 2013.

```
<mbean code="org.jboss.resource.deployment.AdminObject"
       name="qpid.jca:name=HelloQueue">
   <attribute name="JNDIName">HelloQueue</attribute>
   <depends optional-attribute-name="RARName">
      jboss.jca:service=RARDeployment,name='qpid-ra-<ra-version>.rar'
   </depends>
   <attribute name="Type">
      org.apache.qpid.ra.admin.QpidQueue
   </attribute>
   <attribute name="Properties">
       DestinationAddress=amq.direct
   </attribute>
</mbean>
```

The above XML defines a JMS Queue which is bound into JNDI as:

```
HelloQueue
```

This destination can be retrieved from JNDI and used for the consumption or production of messages. The ***`DestinationAddress`*** property can be customized for your environment. Refer to Qpid Java Client documentation for specific configuration options.

Report a bug

11.5.2.3.3. JMS Topic Administered Object Example

Changes

- Updated April 2013.

```
<mbean code="org.jboss.resource.deployment.AdminObject"
       name="qpid.jca:name=HelloTopic">
   <attribute name="JNDIName">
      HelloTopic
   </attribute>
   <depends optional-attribute-name="RARName">
      jboss.jca:service=RARDeployment,name='qpid-ra-<ra-version>.rar'
   </depends>
   <attribute name="Type">
      org.apache.qpid.ra.admin.QpidTopic
   </attribute>
   <attribute name="Properties">
       DestinationAddress=amq.topic
   </attribute>
</mbean>
```

The above XML defines a JMS Topic which is bound into JNDI as:

```
HelloTopic
```

This destination can be retrieved from JNDI and used for the consumption or production of messages. The ***`DestinationAddress`*** property can be customized for your environment. Refer to Qpid Java Client documentation for specific configuration options.

Report a bug

11.5.2.3.4. ConnectionFactory Administered Object Example

```
<mbean code="org.jboss.resource.deployment.AdminObject"
       name="qpid.jca:name=QpidConnectionFactory">
   <attribute name="JNDIName">
      QpidConnectionFactory
   </attribute>
   <depends optional-attribute-name="RARName">
      jboss.jca:service=RARDeployment,name='qpid-ra-<ra-version>.rar'
   </depends>
   <attribute name="Type">
      javax.jms.ConnectionFactory
   </attribute>
   <attribute name="Properties">
      ConnectionURL=amqp://anonymous:@client/test?
brokerlist='tcp://localhost:5672?sasl_mechs='ANONYMOUS''
   </attribute>
</mbean>
```

The above XML defines a ConnectionFactory that can be used for JBoss EAP 5 and also other external clients. Typically, this connection factory is used by standalone or 'thin' clients which do not require an application server. This object is bound into the JBoss EAP 5 JNDI tree as:

```
QpidConnectionFactory
```

Report a bug

11.6. Deploying the Qpid JCA Adapter on JBoss EAP 6

11.6.1. Deploy the Qpid JCA Adapter on JBoss EAP 6

To install the Qpid JCA Adapter for use with JBoss EAP 6, copy the Qpid JCA Adapter configuration files to the JBoss deploy directory.

Procedure 11.2. To deploy the Qpid JCA adapter for JBoss EAP

1. Locate the **qpid-ra-<*version*>.rar** file. It is a zip archive data file that contains the resource adapter, Qpid Java client **.jar** files and the **META-INF** directory.
2. Copy the **qpid-ra-<*version*>.rar** file to your JBoss deploy directory. The JBoss deploy directory is ***JBOSS_ROOT*/<*server-config*>/deployments**, where *JBOSS_ROOT* is the root directory of your JBoss installation and <*server-config*> is the name of your deployment server configuration.

When the JCA Adapter is deployed, it must be configured before it can be used.

Report a bug

11.6.2. JCA Configuration on JBoss EAP 6

11.6.2.1. JCA Adapter Configuration Files in JBoss EAP 6

JBoss EAP 6.x uses a different configuration scheme from previous EAP versions.

Each server configuration type contains the following configuration files in the directory ***JBOSS_ROOT*/<*server-config*>/configuration**:

- <*server-config*>-full.xml
- <*server-config*>-full-ha.xml
- <*server-config*>.xml

Each file corresponds to a set of capabilities, and contains the configuration of the subsystems for that server profile.

Report a bug

11.6.2.2. Replace the Default Messaging Provider with the Qpid JCA Adapter

The following XML fragment from a server configuration file replaces the default EAP 6 messaging provider:

```
<subsystem xmlns="urn:jboss:domain:ejb3:1.2">
   <session-bean>
      <stateless>
         <bean-instance-pool-ref pool-name="slsb-strict-max-pool"/>
      </stateless>
      <stateful default-access-timeout="5000" cache-ref="simple"/>
      <singleton default-access-timeout="5000"/>
   </session-bean>
   <mdb>
      <resource-adapter-ref resource-adapter-name="qpid-ra-<rar-
version>.rar"/>
      <bean-instance-pool-ref pool-name="mdb-strict-max-pool"/>
   </mdb>
```

```
   <pools>
      <bean-instance-pools>
         <strict-max-pool name="slsb-strict-max-pool" max-pool-size="20"
instance-acquisition-timeout="5" instance-acquisition-timeout-
unit="MINUTES"/>
         <strict-max-pool name="mdb-strict-max-pool" max-pool-size="20"
instance-acquisition-timeout="5" instance-acquisition-timeout-
unit="MINUTES"/>
      </bean-instance-pools>
   </pools>
   <caches>
      <cache name="simple" aliases="NoPassivationCache"/>
      <cache name="passivating" passivation-store-ref="file"
aliases="SimpleStatefulCache"/>
   </caches>
   <passivation-stores>
      <file-passivation-store name="file"/>
   </passivation-stores>
   <async thread-pool-name="default"/>
   <timer-service thread-pool-name="default">
      <data-store path="timer-service-data" relative-
to="jboss.server.data.dir"/>
   </timer-service>
   <remote connector-ref="remoting-connector" thread-pool-name="default"/>
   <thread-pools>
      <thread-pool name="default">
         <max-threads count="10"/>
         <keepalive-time time="100" unit="milliseconds"/>
      </thread-pool>
   </thread-pools>
</subsystem>
```

The relevant lines in this sub-system configuration are:

```
<mdb>
   <resource-adapter-ref resource-adapter-name="qpid-ra-<rar-version>.rar"/>
   <bean-instance-pool-ref pool-name="mdb-strict-max-pool"/>
</mdb>
```

Report a bug

11.6.2.3. Configuration Methods

You have two options:

1. Directly edit the existing configuration file.
2. Copy the existing configuration file, edit the copy, then start the server using the new configuration file with the command:

```
JBOSS_HOME/bin/standalone.sh -c your-modified-config.xml
```

Report a bug

11.6.2.4. Example Minimal EAP 6 Configuration

The following XML fragment from a JBoss EAP 6 server configuration file is a minimal example configuration, configuring an XA aware ManagedConnectionFactory and two JMS destinations (queue and topic)

```
<subsystem xmlns="urn:jboss:domain:resource-adapters:1.0">
    <resource-adapters>
        <resource-adapter>
            <archive>
                qpid-ra-<rar-version>.rar
            </archive>
            <transaction-support>
                XATransaction
            </transaction-support>
            <config-property name="connectionURL">
                amqp://anonymous:passwd@client/test?
brokerlist='tcp://localhost?sasl_mechs='PLAIN''
            </config-property>
            <config-property name="TransactionManagerLocatorClass">
                org.apache.qpid.ra.tm.JBoss7TransactionManagerLocator
            </config-property>
            <config-property name="TransactionManagerLocatorMethod">
                getTm
            </config-property>
            <connection-definitions>
                <connection-definition class-
name="org.apache.qpid.ra.QpidRAManagedConnectionFactory" jndi-
name="QpidJMSXA" pool-name="QpidJMSXA">
                    <config-property name="connectionURL">
                        amqp://anonymous:passwd@client/test?
brokerlist='tcp://localhost?sasl_mechs='PLAIN''
                    </config-property>
                    <config-property name="SessionDefaultType">
                        javax.jms.Queue
                    </config-property>
                </connection-definition>
            </connection-definitions>
            <admin-objects>
                <admin-object class-
name="org.apache.qpid.ra.admin.QpidTopicImpl" jndi-
name="java:jboss/exported/GoodByeTopic" use-java-context="false" pool-
name="GoodByeTopic">
                    <config-property name="DestinationAddress">
                        amq.topic/hello.Topic
                    </config-property>
                </admin-object>
                <admin-object class-
name="org.apache.qpid.ra.admin.QpidQueueImpl" jndi-
name="java:jboss/exported/HelloQueue" use-java-context="false" pool-
name="HelloQueue">
                    <config-property name="DestinationAddress">
                        hello.Queue;{create:always, node:{type:queue, x-
declare:{auto-delete:true}}}
                    </config-property>
                </admin-object>
```

```
            </admin-objects>
        </resource-adapter>
    </resource-adapters>
</subsystem>
```

Report a bug

11.6.2.5. Further Resources

For further information, refer to the JBoss Enterprise Application Platform documentation and the README files included with the Qpid JCA Adapter.

Report a bug

Chapter 12. Management Tools and Consoles

12.1. Command-line utilities

12.1.1. Command-line Management utilities

The command-line tools are management and diagnostic tools for use at the shell prompt.

Table 12.1. Command-line Management utilities

Utility	Description
qpid-config	Display and configure exchanges, queues, and bindings in the broker
qpid-tool	Access configuration, statistics, and control within the broker
qpid-queue-stats	Monitor the size and enqueue/dequeue rates of queues in a broker
qpid-ha	Configure and view clusters
qpid-route	Configure federated routes among brokers
qpid-stat	Display details and statistics for various broker objects
qpid-printevents	Subscribes to events from a broker and prints details of events raised to the console window

Report a bug

12.1.2. Using qpid-config

1. View the full list of commands by running the **qpid-config --help** command from the shell prompt:

```
$ qpid-config --help

Usage:  qpid-config [OPTIONS]
qpid-config [OPTIONS] exchanges [filter-string]
qpid-config [OPTIONS] queues    [filter-string]
qpid-config [OPTIONS] add exchange <type> <name> [AddExchangeOptions]
qpid-config [OPTIONS] del exchange <name>
..[output truncated]...
```

2. View a summary of all exchanges and queues by using the **qpid-config** without options:

```
$ qpid-config

Total Exchanges: 6
   topic: 2
 headers: 1
  fanout: 1
  direct: 2
   Total Queues: 7
 durable: 0
    non-durable: 7
```

3. List information on all existing queues by using the **queues** command:

```
$ qpid-config queues
Queue Name                                      Attributes
================================================================
my-queue                                        --durable
qmfc-v2-hb-localhost.localdomain.20293.1  auto-del excl --limit-
policy=ring
qmfc-v2-localhost.localdomain.20293.1     auto-del excl
qmfc-v2-ui-localhost.localdomain.20293.1  auto-del excl --limit-
policy=ring
reply-localhost.localdomain.20293.1       auto-del excl
topic-localhost.localdomain.20293.1       auto-del excl --limit-
policy=ring
```

4. Add new queues with the **add queue** command and the name of the queue to create:

```
$ qpid-config add queue queue_name
```

5. To delete a queue, use the **del queue** command with the name of the queue to remove:

```
$ qpid-config del queue queue_name
```

6. List information on all existing exchanges with the **exchanges** command. Add the **-r** option ("recursive") to also see binding information:

```
$ qpid-config -r exchanges

Exchange '' (direct)
 bind pub_start => pub_start
 bind pub_done => pub_done
 bind sub_ready => sub_ready
 bind sub_done => sub_done
 bind perftest0 => perftest0
 bind mgmt-3206ff16-fb29-4a30-82ea-e76f50dd7d15 => mgmt-3206ff16-fb29-
4a30-82ea-e76f50dd7d15
 bind repl-3206ff16-fb29-4a30-82ea-e76f50dd7d15 => repl-3206ff16-fb29-
4a30-82ea-e76f50dd7d15
Exchange 'amq.direct' (direct)
 bind repl-3206ff16-fb29-4a30-82ea-e76f50dd7d15 => repl-3206ff16-fb29-
4a30-82ea-e76f50dd7d15
 bind repl-df06c7a6-4ce7-426a-9f66-da91a2a6a837 => repl-df06c7a6-4ce7-
426a-9f66-da91a2a6a837
 bind repl-c55915c2-2fda-43ee-9410-b1c1cbb3e4ae => repl-c55915c2-2fda-
43ee-9410-b1c1cbb3e4ae
Exchange 'amq.topic' (topic)
Exchange 'amq.fanout' (fanout)
Exchange 'amq.match' (headers)
Exchange 'qpid.management' (topic)
 bind mgmt.# => mgmt-3206ff16-fb29-4a30-82ea-e76f50dd7d15
```

7. Add new exchanges with the **add exchange** command. Specify the type (direct, topic or fanout) along with the name of the exchange to create. You can also add the **--durable** option to make the exchange durable:

```
$ qpid-config add exchange direct exchange_name --durable
```

8. To delete an exchange, use the **del exchange** command with the name of the exchange to remove:

```
$ qpid-config del exchange exchange_name
```

Report a bug

12.1.3. Using qpid-tool

1. The **qpid-tool** creates a connection to a broker, and commands are run within the tool, rather than at the shell prompt itself. To create the connection, run the **qpid-tool** at the shell prompt with the name or IP address of the machine running the broker you wish to view. You can also append a TCP port number with a **:** character:

```
$ qpid-tool localhost

Management Tool for QPID
qpid:
```

2. If the connection is successful, qpid-tool will display a **qpid:** prompt. Type **help** at this prompt to see the full list of commands:

```
qpid: help
Management Tool for QPID

Commands:
list                    - Print summary of existing objects by class
list <className>        - Print list of objects of the specified class
list <className> all    - Print contents of all objects of specified
class
...[output truncated}...
```

3. **qpid-tool** uses the word *objects* to refer to queues, exchanges, brokers and other such devices. To view a list of all existing objects, type **list** at the prompt:

```
# qpid-tool
Management Tool for QPID
qpid: list
Summary of Objects by Type:
    Package                 Class         Active  Deleted
    ================================================================
    org.apache.qpid.broker  exchange      8       0
    org.apache.qpid.broker  broker        1       0
    org.apache.qpid.broker  binding       16      12
    org.apache.qpid.broker  session       2       1
    org.apache.qpid.broker  connection    2       1
```

```
    org.apache.qpid.broker  vhost         1       0
    org.apache.qpid.broker  queue         6       5
    org.apache.qpid.broker  system        1       0
    org.apache.qpid.broker  subscription  6       5
```

4. You can choose which objects to list by also specifying a class:

```
qpid: list system
Object Summary:
    ID   Created   Destroyed  Index

=====================================================================
==
    167  07:34:13  -          UUID('b3e2610e-5420-49ca-8306-
dca812db647f')
```

5. To view details of an object class, use the **schema** command and specify the class:

```
qpid: schema queue
Schema for class 'qpid.queue':
Element                Type          Unit         Access      Notes
Description
=====================================================================
==========================================
vhostRef               reference                  ReadCreate  index
name                   short-string               ReadCreate  index
durable                boolean                    ReadCreate
autoDelete             boolean                    ReadCreate
exclusive              boolean                    ReadCreate
arguments              field-table                ReadOnly
Arguments supplied in queue.declare
storeRef               reference                  ReadOnly
Reference to persistent queue (if durable)
msgTotalEnqueues       uint64        message
Total messages enqueued
msgTotalDequeues       uint64        message
Total messages dequeued
msgTxnEnqueues         uint64        message
Transactional messages enqueued
msgTxnDequeues         uint64        message
Transactional messages dequeued
msgPersistEnqueues     uint64        message
Persistent messages enqueued
msgPersistDequeues     uint64        message
Persistent messages dequeued
msgDepth               uint32        message
Current size of queue in messages
msgDepthHigh           uint32        message
Current size of queue in messages (High)
msgDepthLow            uint32        message
Current size of queue in messages (Low)
byteTotalEnqueues      uint64        octet
Total messages enqueued
byteTotalDequeues      uint64        octet
Total messages dequeued
```

```
byteTxnEnqueues        uint64        octet
Transactional messages enqueued
byteTxnDequeues        uint64        octet
Transactional messages dequeued
bytePersistEnqueues    uint64        octet
Persistent messages enqueued
bytePersistDequeues    uint64        octet
Persistent messages dequeued
byteDepth              uint32        octet
Current size of queue in bytes
byteDepthHigh          uint32        octet
Current size of queue in bytes (High)
byteDepthLow           uint32        octet
Current size of queue in bytes (Low)
enqueueTxnStarts       uint64        transaction
Total enqueue transactions started
enqueueTxnCommits      uint64        transaction
Total enqueue transactions committed
enqueueTxnRejects      uint64        transaction
Total enqueue transactions rejected
enqueueTxnCount        uint32        transaction
Current pending enqueue transactions
enqueueTxnCountHigh    uint32        transaction
Current pending enqueue transactions (High)
enqueueTxnCountLow     uint32        transaction
Current pending enqueue transactions (Low)
dequeueTxnStarts       uint64        transaction
Total dequeue transactions started
dequeueTxnCommits      uint64        transaction
Total dequeue transactions committed
dequeueTxnRejects      uint64        transaction
Total dequeue transactions rejected
dequeueTxnCount        uint32        transaction
Current pending dequeue transactions
dequeueTxnCountHigh    uint32        transaction
Current pending dequeue transactions (High)
dequeueTxnCountLow     uint32        transaction
Current pending dequeue transactions (Low)
consumers              uint32        consumer
Current consumers on queue
consumersHigh          uint32        consumer
Current consumers on queue (High)
consumersLow           uint32        consumer
Current consumers on queue (Low)
bindings               uint32        binding
Current bindings
bindingsHigh           uint32        binding
Current bindings (High)
bindingsLow            uint32        binding
Current bindings (Low)
unackedMessages        uint32        message
Messages consumed but not yet acked
unackedMessagesHigh    uint32        message
Messages consumed but not yet acked (High)
unackedMessagesLow     uint32        message
Messages consumed but not yet acked (Low)
```

```
messageLatencySamples  delta-time    nanosecond
Broker latency through this queue (Samples)
messageLatencyMin      delta-time    nanosecond
Broker latency through this queue (Min)
messageLatencyMax      delta-time    nanosecond
Broker latency through this queue (Max)
messageLatencyAverage  delta-time    nanosecond
Broker latency through this queue (Average)
```

6. To exit the tool and return to the shell, type **quit** at the prompt:

```
qpid: quit
Exiting...
```

Report a bug

12.1.4. Using qpid-queue-stats

Run the command **qpid-queue-stats** to launch the tool.

The tool will begin to report stats for the broker on the current machine, with the following format:

```
Queue Name                                       Sec       Depth     Enq
Rate     Deq Rate
================================================================================
============
message_queue                                  10.00      11224
0.00        54.01
qmfc-v2-ui-radhe.26001.1                       10.00          0
0.10         0.10
topic-radhe.26001.1                            10.00          0
0.20         0.20
message_queue                                  10.01       9430
0.00       179.29
qmfc-v2-ui-radhe.26001.1                       10.01          0
0.10         0.10
topic-radhe.26001.1                            10.01          0
0.20         0.20
```

The queues on the broker are listed on the left. The *Sec* column is the sample rate. The tool is hard-coded to poll the broker in 10 second intervals. The *Depth* column reports the number of messages in the queue. *Enq Rate* is the number of messages added to the queue (enqueued) since the last sample. *Deq Rate* is the number of messages removed from the queue (dequeued) since the last sample.

To view the queues on another server, use the **-a** switch, and provide a remote server address, and optionally the remote port and authentication credentials.

For example, to examine the queues on a remote server with the address **192.168.1.145**, issue the command:

```
qpid-queue-stats -a 192.168.1.145
```

To examine the queues on the server **broker1.mydomain.com**:

```
qpid-queue-stats -a broker1.mydomain.com
```

To examine the queues on the server **broker1.mydomain.com**, where the broker is running on port 8888:

```
qpid-queue-stats -a broker1.mydomain.com:8888
```

To examine the queues on the server **192.168.1.145**, which requires authentication:

```
qpid-queue-stats -a username/password@192.168.1.145
```

Report a bug

Appendix A. Exchange and Queue Declaration Arguments

A.1. Exchange and Queue Argument Reference

Changes

- **`qpid.last_value_queue`** and **`qpid.last_value_queue_no_browse`** deprecated and removed.
- **`qpid.msg_sequence`** queue argument replaced by **`qpid.queue_msg_sequence`**.
- **`ring_strict`** and **`flow_to_disk`** are no longer valid **`qpid.policy_type`** values.
- **`qpid.persist_last_node`** deprecated and removed.

Following is a complete list of arguments for declaring queues and exchanges.

Exchange options

`qpid.exclusive-binding` (bool)

Ensures that a given binding key is associated with only one queue.

`qpid.ive` (bool)

If set to “true”, the exchange is an *initial value exchange*, which differs from other exchanges in only one way: the last message sent to the exchange is cached, and if a new queue is bound to the exchange, it attempts to route this message to the queue, if the message matches the binding criteria. This allows a new queue to use the last received message as an initial value.

`qpid.msg_sequence` (bool)

If set to “true”, the exchange inserts a sequence number named “qpid.msg_sequence” into the message headers of each message. The type of this sequence number is int64. The sequence number for the first message routed from the exchange is 1, it is incremented sequentially for each subsequent message. The sequence number is reset to 1 when the qpid broker is restarted.

`qpid.sequence_counter` (int64)

Start **`qpid.msg_sequence`** counting at the given number.

Queue options

`no-local` (bool)

Specifies that the queue should discard any messages enqueued by sessions on the same connection as that which declares the queue.

`qpid.alert_count` (uint32_t)

If the queue message count goes above this size an alert should be sent.

`qpid.alert_repeat_gap` (int64_t)

Controls the minimum interval between events in seconds. The default value is 60 seconds.

`qpid.alert_size` (int64_t)

If the queue size in bytes goes above this size an alert should be sent.

`qpid.auto_delete_timeout` (bool)

If a queue is configured to be automatically deleted, it will be deleted after the amount of seconds specified here.

`qpid.browse-only` (bool)

All users of queue are forced to browse. Limit queue size with ring, LVQ, or TTL. Note that this argument name uses a hyphen rather than an underscore.

`qpid.file_count` (int)

Set the number of files in the persistence journal for the queue. Default value is 8.

`qpid.file_size` (int64)

Set the number of pages in the file (each page is 64KB). Default value is 24.

`qpid.flow_resume_count` (uint32_t)

Flow resume threshold value as a message count.

`qpid.flow_resume_size` (uint64_t)

Flow resume threshold value in bytes.

`qpid.flow_stop_count` (uint32_t)

Flow stop threshold value as a message count.

`qpid.flow_stop_size` (uint64_t)

Flow stop threshold value in bytes.

`qpid.last_value_queue_key` (string)

Defines the key to use for a last value queue.

`qpid.max_count` (uint32_t)

The maximum byte size of message data that a queue can contain before the action dictated by the **`policy_type`** is taken.

`qpid.max_size` (uint64_t)

The maximum number of messages that a queue can contain before the action dictated by the **`policy_type`** is taken.

`qpid.policy_type` (string)

Sets default behavior for controlling queue size. Valid values are ***`reject`*** and ***`ring`***.

`qpid.priorities` (size_t)

The number of distinct priority levels recognized by the queue (up to a maximum of 10). The default value is 1 level.

`qpid.queue_msg_sequence` (string)

Causes a custom header with the specified name to be added to enqueued messages. This header is automatically populated with a sequence number.

`qpid.trace.exclude` (string)

Does not send on messages which include one of the given (comma separated) trace ids.

`qpid.trace.id` (string)

Adds the given trace id as to the application header "**`x-qpid.trace`**" in messages sent from the queue.

`x-qpid-maximum-message-count`

This is an alias for **`qpid.alert_count`**.

`x-qpid-maximum-message-size`

This is an alias for **`qpid.alert_size`**.

`x-qpid-minimum-alert-repeat-gap`

This is an alias for **`qpid.alert_repeat_gap`**.

`x-qpid-priorities`

This is an alias for **`qpid.priorities`**.

Report a bug

Appendix B. OpenSSL Certificate Reference

B.1. Reference of Certificates

This reference for creating and managing certificates with the **openssl** command assumes familiarity with SSL. For more background information on SSL refer to the OpenSSL documentation at www.openssl.org.

Important

It is recommended that only certificates signed by an authentic Certificate Authority (CA) are used for secure systems. Instructions in this section for generating self-signed certificates are meant to facilitate test and development activities or evaluation of software while waiting for a certificate from an authentic CA.

Generating Certificates

Procedure B.1. Create a Private Key

- Use this command to generate a 1024-bit RSA private key with file encryption. If the key file is encrypted, the password will be needed every time an application accesses the private key.

```
# openssl genrsa -des3 -out mykey.pem 1024
```

Use this command to generate a key without file encryption:

```
# openssl genrsa  -out mykey.pem 1024
```

Procedure B.2. Create a Self-Signed Certificate

Each of the following commands generates a new private key and a *self-signed* certificate, which acts as its own CA and does not need additional signatures. This certificate expires one week from the time it is generated.

1. The ***nodes*** option causes the key to be stored without encryption. OpenSSL will prompt for values needed to create the certificate.

```
# openssl req -x509 -nodes -days 7 -newkey rsa:1024 -keyout mykey.pem -
out mycert.pem
```

2. The ***subj*** option can be used to specify values and avoid interactive prompts, for example:

```
# openssl req -x509 -nodes -days 7 -subj
'/C=US/ST=NC/L=Raleigh/CN=www.redhat.com' -newkey rsa:1024 -keyout
mykey.pem -out mycert.pem
```

3. The ***new*** and ***key*** options generate a certificate using an existing key instead of generating a new one.

```
# openssl req -x509 -nodes -days 7 -new -key mykey.pem -out mycert.pem
```

Create a Certificate Signing Request

To generate a certificate and have it signed by a Certificate Authority (CA), you need to generate a certificate signing request (CSR):

```
# openssl req -new -key mykey.pem -out myreq.pem
```

The certificate signing request can now be sent to an authentic Certificate Authority for signing and a valid signed certificate will be returned. The exact procedure to send the CSR and receive the signed certificate depend on the particular Certificate Authority you use.

Create Your Own Certificate Authority

You can create your own Certificate Authority and use it to sign certificate requests. If the Certificate Authority is added as a trusted authority on a system, any certificates signed by the Certificate Authority will be valid on that system. This option is useful if a large number of certificates are needed temporarily.

1. Create a self-signed certificate for the CA, as described in Procedure B.2, “Create a Self-Signed Certificate”.
2. OpenSSL needs the following files set up for the CA to sign certificates. On a Red Hat Enterprise Linux system with a fresh OpenSSL installation using a default configuration, set up the following files:
    a. Set the path for the CA certificate file as **/etc/pki/CA/cacert.pem**.
    b. Set the path for the CA private key file as **/etc/pki/CA/private/cakey.pem**.
    c. Create a zero-length index file at **/etc/pki/CA/index.txt**.
    d. Create a file containing an initial serial number (for example, 01) at **/etc/pki/CA/serial**.
    e. The following steps must be performed on RHEL 5:
        i. Create the directory where new certificates will be stored: **/etc/pki/CA/newcerts**.
        ii. Change to the certificate directory: **cd /etc/pki/tls/certs**.
3. The following command signs a CSR using the CA:

```
# openssl ca -notext -out mynewcert.pem -infiles myreq.pem
```

Install a Certificate

1. For OpenSSL to recognize a certificate, a hash-based symbolic link must be generated in the **certs** directory. **/etc/pki/tls** is the parent of the **certs** directory in Red Hat Enterprise Linux's version of OpenSSL. Use the **version** command to check the parent directory:

```
# openssl version -d
OPENSSLDIR: "/etc/pki/tls"
```

2. Create the required symbolic link for a certificate using the following command:

```
# ln -s certfile `openssl x509 -noout -hash -in certfile`.0
```

It is possible for more than one certificate to have the same hash value. If this is the case, change the suffix on the link name to a higher number. For example:

```
# ln -s certfile `openssl x509 -noout -hash -in certfile`.4
```

Examine Values in a Certificate

The content of a certificate can be seen in plain text with this command:

```
# openssl x509 -text -in mycert.pem
```

Exporting a Certificate from NSS into PEM Format

Certificates stored in an NSS certificate database can be exported and converted to PEM format in several ways:

- This command exports a certificate with a specified nickname from an NSS database:

  ```
  # certutil -d . -L -n "Some Cert" -a > somecert.pem
  ```

- These commands can be used together to export certificates and private keys from an NSS database and convert them to PEM format. They produce a file containing the client certificate, the certificate of its CA, and the private key.

  ```
  # pk12util -d . -n "Some Cert" -o somecert.pk12
  # openssl pkcs12 -in somecert.pk12 -out tmckay.pem
  ```

  See documentation for the **openssl pkcs12** command for options that limit the content of the PEM output file.

Report a bug

Appendix C. Revision History

Revision 3.2.0-7	**June 2017**	**Susan Jay**
Post-GA update.		
Revision 3.2.0-6	**Fri Oct 16 2015**	**Scott Mumford**
Post-GA update.		
Revision 3.2.0-5	**Thu Oct 8 2015**	**Scott Mumford**
MRG-M 3.2 GA		
Revision 3.2.0-3	**Tue Sep 29 2015**	**Scott Mumford**
Prepared for MRG-M 3.2 GA		
Revision 3.2.0-1	**Tue Jul 14 2015**	**Jared Morgan**
Prepared for MRG-M 3.2 GA		
Revision 3.1.0-5	**Wed Apr 01 2015**	**Jared Morgan**
Prepared for MRG-M 3.1 GA		
Revision 3.0.0-1	**Tue Sep 23 2014**	**Jared Morgan**
Prepared for MRG-M 3.0 GA		