AMQP Messaging Broker (Java)

AMQP Messaging Broker (Java)

Table of Contents

List of Figures

List of Tables

List of Examples

Chapter 1. Introduction

The Java Broker is a powerful open-source message broker that implements all versions of the Advanced Message Queuing Protocol (AMQP) [http://www.amqp.org]. The Java Broker is actually one of two message brokers provided by the Apache Qpid project [http://qpid.apache.org]: the Java Broker and the C++ Broker.

This document relates to the Java Broker. The C++ Broker is described separately [../../AMQP-Messaging-Broker-CPP-Book/html/].

Headline features

- 100% Java implementation - runs on any platform supporting Java 1.6 or higher
- Messaging clients support in Java, C++, Python.
- JMS 1.1 compliance (Java client).
- Persistent and non-persistent (transient) message support
- Supports for all common messaging patterns (point-to-point, publish/subscribe, fan-out etc).
- Transaction support including XA[1]
- Supports for all versions of the AMQP protocol
- Automatic message translation, allowing clients using different AMQP versions to communicate with each other.
- Pluggable authentication architecture with out-of-the-box support for Kerberos, LDAP, External, and file-based authentication mechanisms.
- Pluggable message store architecture with implementations based on Apache Derby [http://db.apache.org/derby/], Oracle BDB JE [http://www.oracle.com/technetwork/products/berkeleydb/overview/index-093405.html][2], and Memory Store
- Web based management interface and programmatic management interfaces via REST and JMX APIs.
- SSL support
- High availability (HA) support.[3]

[1]XA provided when using AMQP 0-10
[2]Oracle BDB JE must be downloaded separately.
[3]HA currently only available to users of the optional BDB JE HA based message store.

Chapter 2. Installation

2.1. Introduction

This document describes how to install the Java Broker on both Windows and UNIX platforms.

2.2. Prerequisites

2.2.1. Java Platform

The Java Broker is an 100% Java implementation and as such it can be used on any operating system supporting Java 1.6 or higher. This includes Linux, Solaris, Mac OS X, and Windows XP/Vista/7/8.

The broker has been tested with Java implementations from both Oracle and IBM. Whatever platform you chose, it is recommended that you ensure it is patched with any critical updates made available from the vendor.

Verify that your JVM is installed properly by following these instructions.

2.2.2. Disk

The Java Broker installation requires approximately 20MB of free disk space.

The Java Broker also requires a working directory. The working directory is used for the message store, that is, the area of the file-system used to record persistent messages whilst they are passing through the Broker. The working directory is also used for the default location of the log file. The size of the working directory will depend on the how the Broker is used.

The performance of the file system hosting the work directory is key to the performance of Broker as a whole. For best performance, choose a device that has low latency and one that is uncontended by other applications.

Be aware that there are additional considerations if you are considering hosting the working directory on NFS. See Chapter 10, *Virtual Host Message Stores* for further details.

2.2.3. Memory

Qpid caches messages on the heap for performance reasons, so in general, the Broker will benefit from as much heap as possible. However, on a 32bit JVM, the maximum addressable memory range for a process is 4GB, after leaving space for the JVM's own use this will give a maximum heap size of approximately ~3.7GB.

2.2.4. Operating System Account

Installation or operation of Qpid does *not* require a privileged account (i.e. root on UNIX platforms or Administrator on Windows). However it is suggested that you use an dedicated account (e.g. qpid) for the installation and operation of the Java Broker.

2.3. Download

2.3.1. Broker Release

You can download the latest Java broker package from the Download Page [http://qpid.apache.org/download.html].

It is recommended that you confirm the integrity of the download by verifying the PGP signature matches that available on the site. Instrutions are given on the download page.

2.3.2. Optional Dependencies

The broker has an optional message store implementations backed by Oracle BDB JE. If you wish to use these stores you will need to provide the optional Oracle BDB JE dependency. For more details, see Section 10.4, “BDB Message Store”

2.4. Installation on Windows

Firstly, verify that your JVM is installed properly by following these instructions.

Now chose a directory for Qpid broker installation. This directory will be used for the Qpid JARs and configuration files. It need not be the same location as the work directory used for the persistent message store or the log file (you will choose this location later). For the remainder this example we will assume that location c:\qpid has been chosen.

Next extract the qpid-java-broker-0.28.zip package into the directory, using either the zip file handling offered by Windows (right click the file and select 'Extract All') or a third party tool of your choice.

The extraction of the broker package will have created a directory qpid-broker-0.28 within c:\qpid

```
 Directory of c:\qpid\qpid-broker-0.28

07/25/2012  11:22 PM                   .
09/30/2012  10:51 AM                   ..
09/30/2012  12:24 AM                   bin
08/21/2012  11:17 PM                   etc
07/25/2012  11:22 PM                   lib
07/20/2012  08:10 PM            65,925 LICENSE
07/20/2012  08:10 PM             3,858 NOTICE
07/20/2012  08:10 PM             1,346 README.txt
```

2.4.1. Setting the working directory

Qpid requires a work directory. This directory is used for the default location of the Qpid log file and is used for the storage of persistent messages. The work directory can be set on the command-line (for the lifetime of the command interpreter), but you will normally want to set the environment variable permanently via the Advanced System Settings in the Control Panel.

```
set QPID_WORK=C:\qpidwork
```

If the directory referred to by QPID_WORK does not exist, the Java Broker will attempt to create it on start-up.

2.4.2. Optional Dependencies

The broker has optional message store implementations backed by Oracle BDB JE. If you wish to use these stores you will need to provide the optional Oracle BDB JE dependency. For more details, see Section 10.4, “BDB Message Store”

2.5. Installation on UNIX platforms

Firstly, verify that your JVM is installed properly by following these instructions.

Now chose a directory for Qpid broker installation. This directory will be used for the Qpid JARs and configuration files. It need not be the same location as the work directory used for the persistent message store or the log file (you will choose this location later). For the remainder this example we will assume that location /usr/local/qpid has been chosen.

Next extract the qpid-java-broker-0.28.tar.gz package into the directory.

```
mkdir /usr/local/qpid
cd /usr/local/qpid
tar xvzf qpid-java-broker-0.28.tar.gz
```

The extraction of the broker package will have created a directory qpid-broker-0.28 within /usr/local/qpid

```
ls -la qpid-broker-0.28/
total 152
drwxr-xr-x   8 qpid  qpid    272 25 Jul 23:22 .
drwxr-xr-x  45 qpid  qpid   1530 30 Sep 10:51 ..
-rw-r--r--@  1 qpid  qpid  65925 20 Jul 20:10 LICENSE
-rw-r--r--@  1 qpid  qpid   3858 20 Jul 20:10 NOTICE
-rw-r--r--@  1 qpid  qpid   1346 20 Jul 20:10 README.txt
drwxr-xr-x  10 qpid  qpid    340 30 Sep 00:24 bin
drwxr-xr-x   9 qpid  qpid    306 21 Aug 23:17 etc
drwxr-xr-x  34 qpid  qpid   1156 25 Jul 23:22 lib
```

2.5.1. Setting the working directory

Qpid requires a work directory. This directory is used for the default location of the Qpid log file and is used for the storage of persistent messages. The work directory can be set on the command-line (for the lifetime of the current shell), but you will normally want to set the environment variable permanently the user's shell profile file (~/.bash_profile for Bash etc).

```
export QPID_WORK=/var/qpidwork
```

If the directory referred to by QPID_WORK does not exist, the Java Broker will attempt to create it on start-up.

2.5.2. Optional Dependencies

The broker has an optional message store implementations backed by Oracle BDB JE. If you wish to use these stores you will need to provide the optional Oracle BDB JE dependency. For more details, see Section 10.4, “BDB Message Store”

Chapter 3. Getting Started

3.1. Introduction

This section describes how to start and stop the Java Broker, and outlines the various command line options.

For additional details about the broker configuration store and related command line arguments see Section 5.1, “Broker Configuration Store”. The broker is fully configurable via its Web Management Console, for details of this see Section 5.2.2, “Web Management Console”.

3.2. Starting/Stopping the broker on Windows

Firstly change to the installation directory used during the installation and ensure that the QPID_WORK environment variable is set.

Now use the **qpid-server.bat** to start the server

```
bin\qpid-server.bat
```

Output similar to the following will be seen:

```
[Broker] BRK-1006 : Using configuration : C:\qpidwork\config.json
[Broker] BRK-1007 : Using logging configuration : C:\qpid\qpid-broker-0.28\etc\log
[Broker] BRK-1001 : Startup : Version: 0.28 Build: 1478262
[Broker] BRK-1010 : Platform : JVM : Oracle Corporation version: 1.7.0_21-b11 OS :
[Broker] BRK-1011 : Maximum Memory : 1,060,372,480 bytes
[Broker] BRK-1002 : Starting : Listening on TCP port 5672
[Broker] MNG-1001 : JMX Management Startup
[Broker] MNG-1002 : Starting : RMI Registry : Listening on port 8999
[Broker] MNG-1002 : Starting : JMX RMIConnectorServer : Listening on port 9099
[Broker] MNG-1004 : JMX Management Ready
[Broker] MNG-1001 : Web Management Startup
[Broker] MNG-1002 : Starting : HTTP : Listening on port 8080
[Broker] MNG-1004 : Web Management Ready
[Broker] BRK-1004 : Qpid Broker Ready
```

The BRK-1004 message confirms that the Broker is ready for work. The MNG-1002 and BRK-1002 confirm the ports to which the Broker is listening (for HTTP/JMX management and AMQP respectively).

To stop the Broker, use Control-C or use the Shutdown MBean from the JMX management plugin.

3.3. Starting/Stopping the broker on Unix

Firstly change to the installation directory used during the installation and ensure that the QPID_WORK environment variable is set.

Now use the **qpid-server** script to start the server:

```
bin\qpid-server
```

Output similar to the following will be seen:

```
[Broker] BRK-1006 : Using configuration : /var/qpidwork/config.json
```

```
[Broker] BRK-1007 : Using logging configuration : /usr/local/qpid/qpid-broker-0.28
[Broker] BRK-1001 : Startup : Version: 0.28 Build: exported
[Broker] BRK-1010 : Platform : JVM : Sun Microsystems Inc. version: 1.6.0_32-b05 O
[Broker] BRK-1011 : Maximum Memory : 1,065,025,536 bytes
[Broker] BRK-1002 : Starting : Listening on TCP port 5672
[Broker] MNG-1001 : Web Management Startup
[Broker] MNG-1002 : Starting : HTTP : Listening on port 8080
[Broker] MNG-1004 : Web Management Ready
[Broker] MNG-1001 : JMX Management Startup
[Broker] MNG-1002 : Starting : RMI Registry : Listening on port 8999
[Broker] MNG-1002 : Starting : JMX RMIConnectorServer : Listening on port 9099
[Broker] MNG-1004 : JMX Management Ready
[Broker] BRK-1004 : Qpid Broker Ready
```

The BRK-1004 message confirms that the Broker is ready for work. The MNG-1002 and BRK-1002 confirm the ports to which the Broker is listening (for HTTP/JMX management and AMQP respectively).

To stop the Broker, use Control-C from the controlling shell, use the **bin/qpid.stop** script, use **kill -TERM <pid>**, or the Shutdown MBean from the JMX management plugin.

3.4. Log file

The Java Broker writes a log file to record both details of its normal operation and any exceptional conditions. By default the log file is written within the log subdirectory beneath the work directory - `$QPID_WORK/log/qpid.log` (UNIX) and `%QPID_WORK%\log\qpid.log` (Windows).

For details of how to control the logging, see Section 12.1, “Log Files”

3.5. Using the command line

The Java Broker understands a number of command line options which may be used to customise the configuration.

For additional details about the broker configuration and related command line arguments see Section 5.1, “Broker Configuration Store”. The broker is fully configurable via its Web Management Console, for details of this see Section 5.2.2, “Web Management Console”.

To see usage information for all command line options, use the `--help` option

```
bin/qpid-server --help

usage: Qpid [-cic <path>] [-h] [-icp <path>] [-l <file>] [-mm] [-mmhttp <port>]
            [-mmjmx <port>] [-mmpass <password>] [-mmqv] [-mmrmi <port>] [-os]
            [-sp <path>] [-st <type>] [-v] [-w <period>]
 -cic <path>                                   Create a copy of the initial confi
 --create-initial-config <path>                file, either to an optionally spec
                                               file path, or as initial-config.js
                                               in the current directory

 -h,                                           Print this message
 --help

 -icp  <path>                                  Set the location of initial JSON c
 --initial-config-path <path>                  to use when creating/overwriting a
```

```
                                               broker configuration store

-l <file>                                      Use the specified log4j xml config
--logconfig <file>                             file. By default looks for a file
                                               etc/log4j.xml in the same director
                                               the configuration file

-mm                                            Start broker in management mode,
--management-mode                              disabling the AMQP ports

-mmhttp <port>                                 Override http management port in
--management-mode-http-port <port>             management mode

-mmjmx                                         Override jmx connector port in
--management-mode-jmx-connector-port <port>    management mode

-mmpass  <password>                            Set the password for the managemen
--management-mode-password <password>          mode user mm_admin

-mmqv                                          Make virtualhosts stay in the quie
--management-mode-quiesce-virtualhosts         state during management mode.

-mmrmi <port>                                  Override jmx rmi registry port in
--management-mode-rmi-registry-port <port>     management mode

-os                                            Overwrite the broker configuration
--overwrite-store                              with the current initial configura

-prop "<name=value>"                           Set a configuration property to us
--config-property "<name=value>"               resolving variables in the broker
                                               configuration store, with format
                                               "name=value"

-sp <path>                                     Use given configuration store loca
--store-path <path>

-st <type>                                     Use given broker configuration sto
--store-type <type>

-v                                             Print the version information and
--version

-w <period>                                    Monitor the log file configuration
--logwatch <period>                            for changes. Units are seconds. Ze
                                               means do not check for changes.
```

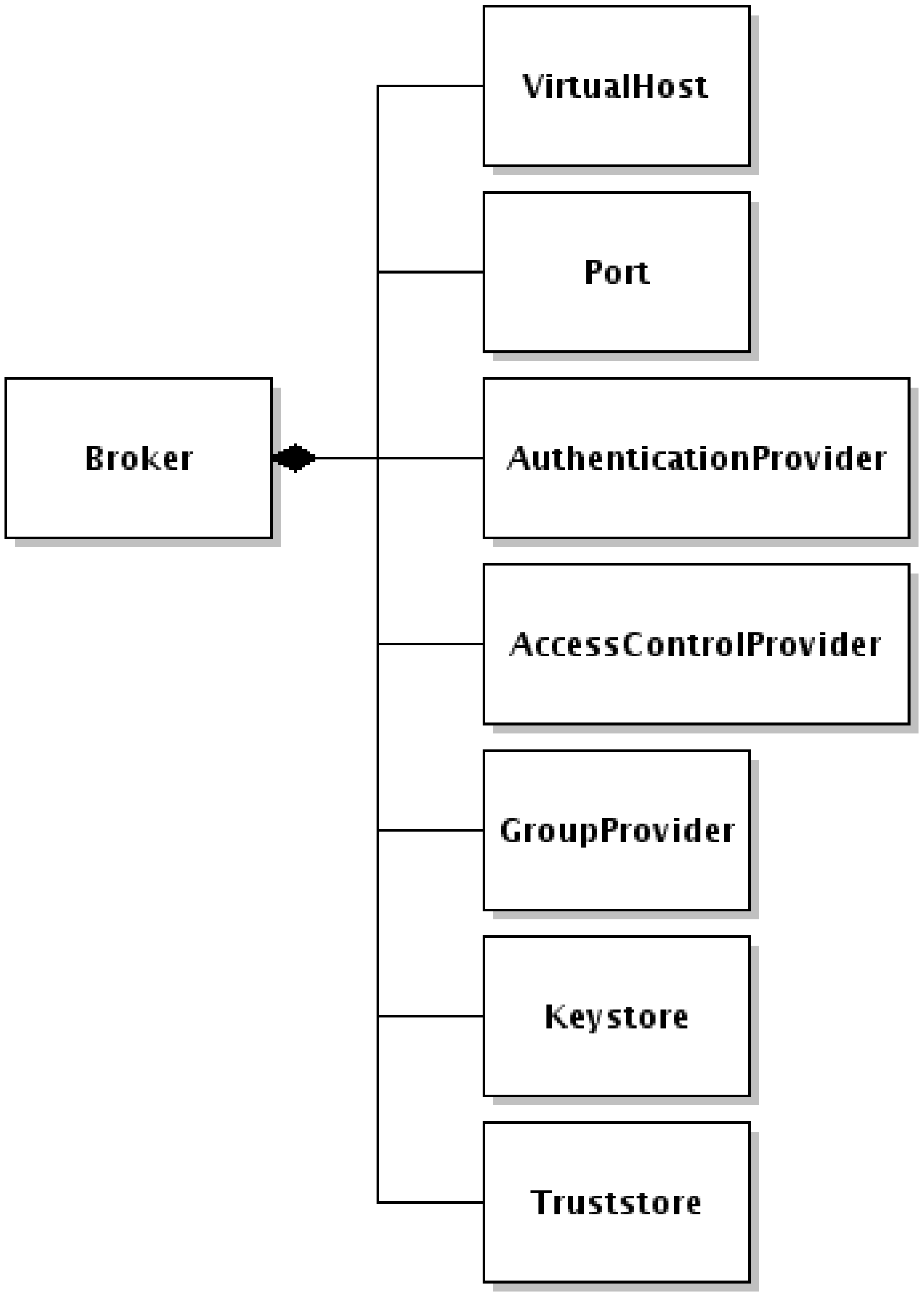
Chapter 4. Concepts

4.1. Broker

The Qpid Broker has one or more *Virtual Hosts* (independent containers of *Queues*, *Exchanges*, etc) sharing a connection, authentication, and access control model via the configured *Ports*, *Authentication Providers*, *Group providers* and *Access Control Providers*. *Keystores* and *Truststores* can be defined on the broker level and linked to *Ports* configured with an SSL transport. The Broker also provides management plugins to allow configuring and monitoring it.

The following diagram depicts the Broker model:

Figure 4.1. Broker Model



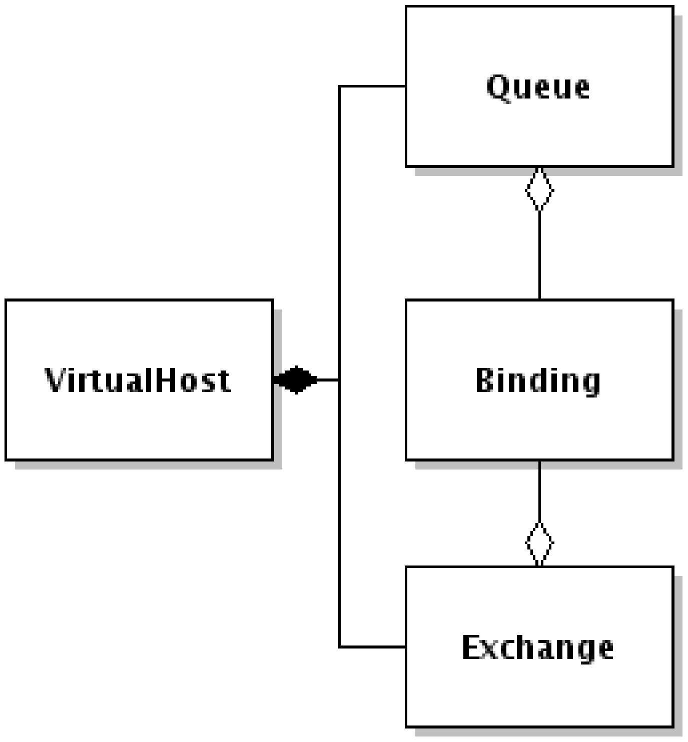
These concepts will be expanded upon in the forthcoming pages.

4.2. Virtual Hosts

A Broker has one or more *Virtual Host*s. Each *Virtual Host* has an independant namespace for its collection of *Exchanges*, *Queues*, and associated objects. Client *Connection*s are made a specific *Virtual Host*, with one being configured as the default for clients that can't or don't specify which they wish to connect to.

The following diagram depicts the Virtual Host model:

Figure 4.2. Virtual Host Model



Each *Virtual Host* has its own *Message Store* which is used to store persistent messages on durable *Queues* it contains, as well as the configuration of any durable *Queues*, *Exchanges*, and *Bindings* made during its operation.

The following message stores are currently supported:

- JDBC Message Store
- Derby Message Store
- Berkeley DB Message Store
- Berkeley DB HA Message Store
- Memory Message Store

Virtual Hosts configuration is covered in Chapter 7, *Virtual Hosts*.

4.3. Exchanges

An *Exchange* is a named entity within the *Virtual Host* which receives messages from producers and routes them to matching *Queue*s within the *Virtual Host*.

The server provides a set of exchange types with each exchange type implementing a different routing algorithm. For details of how these exchanges types work see Section 4.3.2, “Exchange Types” below.

The server predeclares a number of exchange instances with names starting with "`amq.`". These are defined in Section 4.3.1, “Predeclared Exchanges”.

Applications can make use the pre-declared exchanges, or they may declare their own. The number of exchanges within a virtual host is limited only by resource constraints.

The behaviour when an exchange is unable to route a message to any queue is defined in Section 4.3.4, “Unrouteable Messages”

Exchange configuration is covered in Chapter 8, *Exchanges*.

4.3.1. Predeclared Exchanges

Each virtual host pre-declares the following exchanges:

- amq.direct (an instance of a direct exchange)
- amq.topic (an instance of a topic exchange)
- amq.fanout (an instance of a fanout exchange)
- amq.match (an instance of a headers exchange)

The conceptual "`default exchange`" always exists, effectively a special instance of direct exchange which uses the empty string as its name. All queues are automatically bound to it upon their creation using the queue name as the binding key, and unbound upon their deletion. It is not possible to manually add or remove bindings within this exchange.

Applications may not declare exchanges with names beginning with "`amq.`". Such names are reserved for system use.

4.3.2. Exchange Types

The following Exchange types are supported.

- Direct
- Topic
- Fanout
- Headers

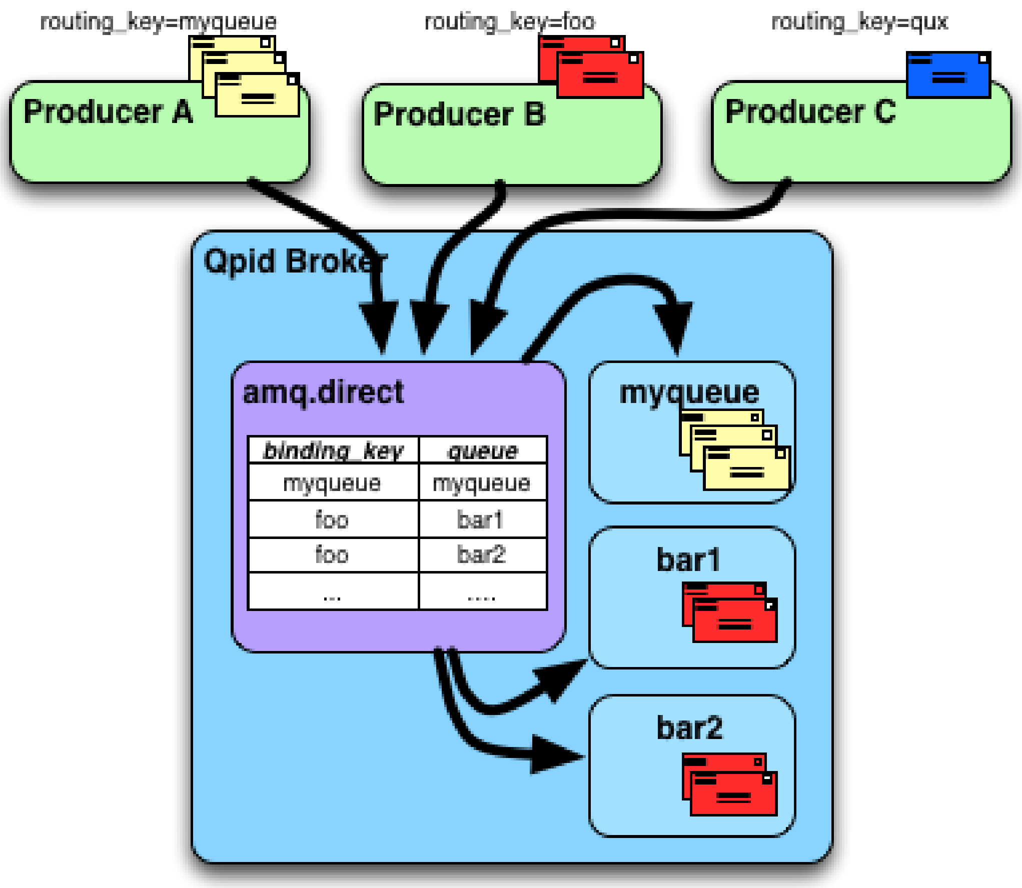
These exchange types are described in the following sub-sections.

4.3.2.1. Direct

The direct exchange type routes messages to queues based on an exact match between the routing key of the message, and the binding key used to bind the queue to the exchange. Additional filter rules may be specified using a binding argument specifying a JMS message selector.

This exchange type is often used to implement point to point messaging. When used in this manner, the normal convention is that the binding key matches the name of the queue. It is also possible to use this exchange type for multi-cast, in this case the same binding key is associated with many queues.

Figure 4.3. Direct exchange



The figure above illustrates the operation of direct exchange type. The yellow messages published with the routing key "`myqueue`" match the binding key corresponding to queue "`myqueue`" and so are routed there. The red messages published with the routing key "`foo`" match two bindings in the table so a copy of the message is routed to both the "`bar1`" and "`bar2`" queues.

The routing key of the blue message matches no binding keys, so the message is unroutable. It is handled as described in Section 4.3.4, "Unrouteable Messages".

4.3.2.2. Topic

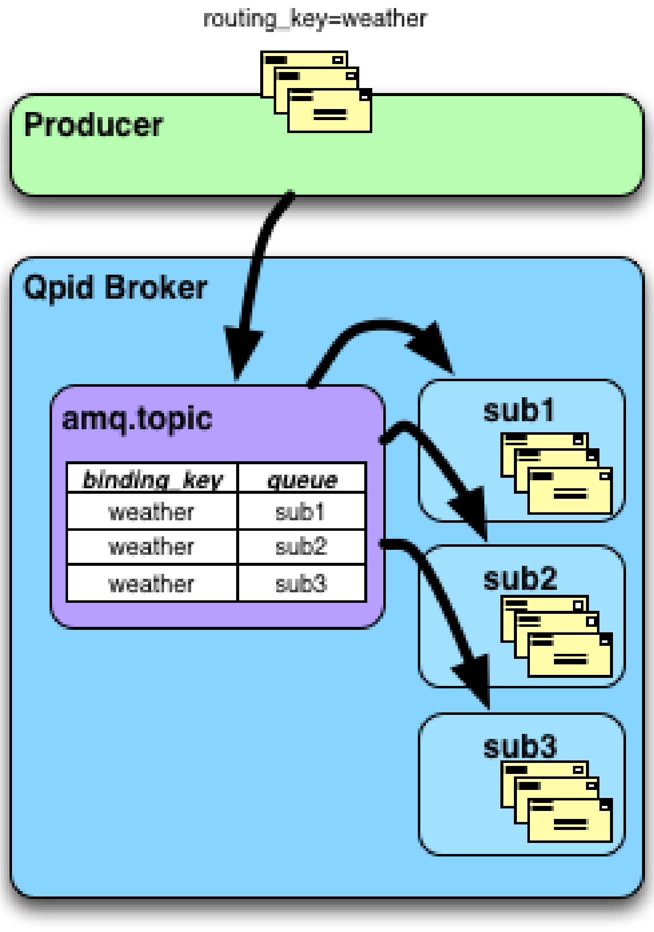
This exchange type is used to support the classic publish/subscribe paradigm.

The topic exchange is capable of routing messages to queues based on wildcard matches between the routing key and the binding key pattern defined by the queue binding. Routing keys are formed from one or more words, with each word delimited by a full-stop (.). The pattern matching characters are the * and # symbols. The * symbol matches a single word and the # symbol matches zero or more words.

Additional filter rules may be specified using a binding argument specifying a JMS message selector.

The following three figures help explain how the topic exchange functions.

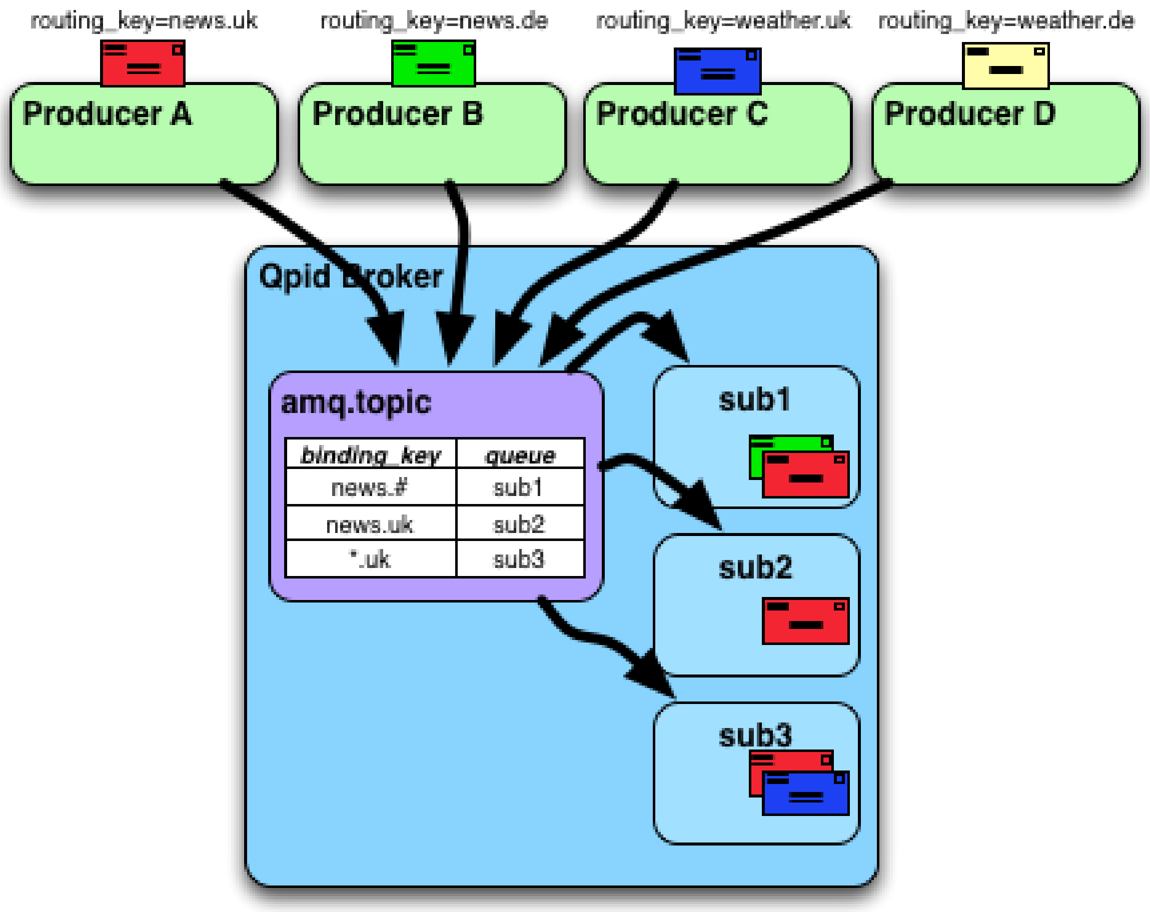
Figure 4.4. Topic exchange - exact match on topic name



The figure above illustrates publishing messages with routing key "`weather`". The exchange routes each message to every bound queue whose binding key matches the routing key.

In the case illustrated, this means that each subscriber's queue receives every yellow message.

Figure 4.5. Topic exchange - matching on hierarchical topic patterns

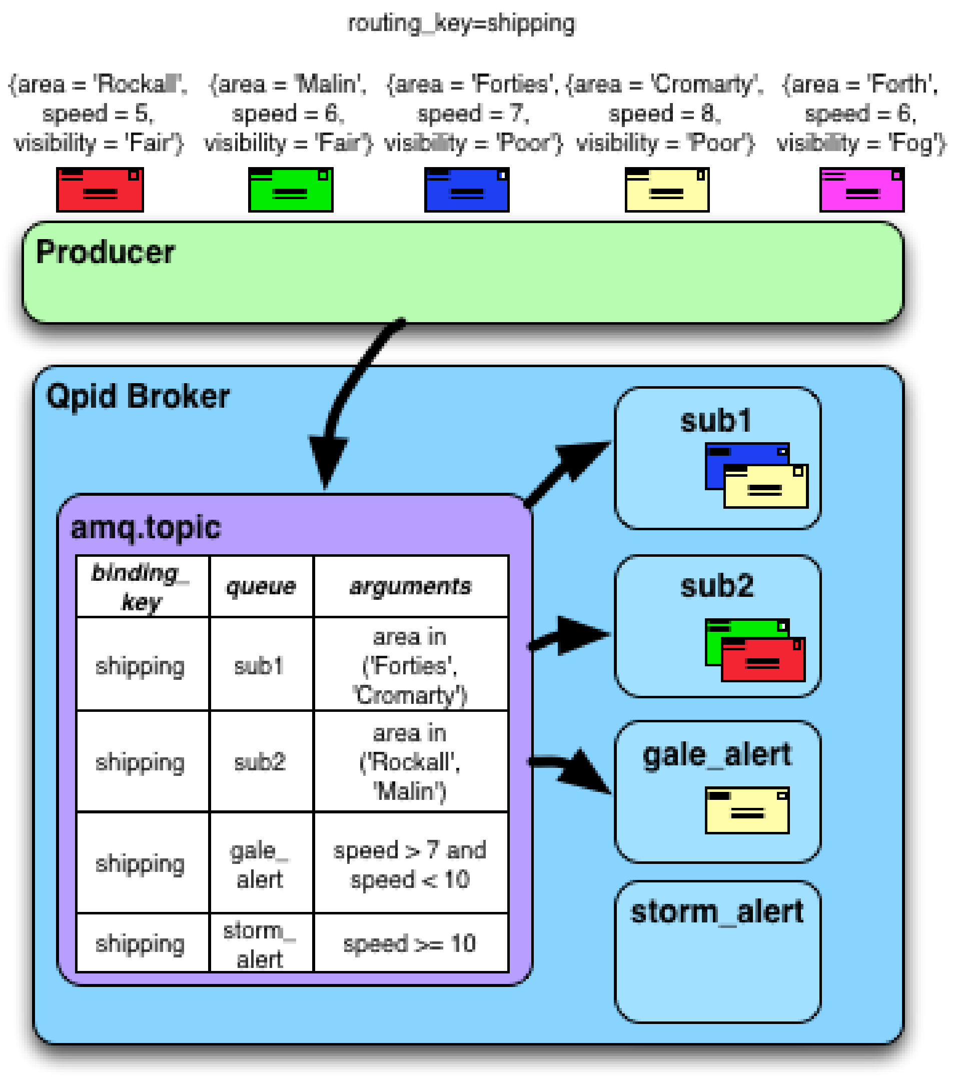


The figure above illustrates publishing messages with hierarchical routing keys. As before, the exchange routes each message to every bound queue whose binding key matches the routing key but as the binding keys contain wildcards, the wildcard rules described above apply.

In the case illustrated, `sub1` has received the red and green message as "`news.uk`" and "`news.de`" match binding key "`news.#`". The red message has also gone to `sub2` and `sub3` as it's routing key is matched exactly by "`news.uk`" and by "`*.uk`".

The routing key of the yellow message matches no binding keys, so the message is unroutable. It is handled as described in Section 4.3.4, “Unrouteable Messages”.

Figure 4.6. Topic exchange - matching on JMS message selector


routing_key=shipping
{area = 'Rockall', speed = 5, visibility = 'Fair'}
{area = 'Malin', speed = 6, visibility = 'Fair'}
{area = 'Forties', speed = 7, visibility = 'Poor'}
{area = 'Cromarty', speed = 8, visibility = 'Poor'}
{area = 'Forth', speed = 6, visibility = 'Fog'}
Producer
Qpid Broker
amq.topic
sub1
sub2
gale_alert
storm_alert

binding_ key	*queue*	*arguments*
shipping	sub1	area in ('Forties', 'Cromarty')
shipping	sub2	area in ('Rockall', 'Malin')
shipping	gale_ alert	speed > 7 and speed < 10
shipping	storm_ alert	speed >= 10

The figure above illustrates messages with properties published with routing key "`shipping`".

As before, the exchange routes each message to every bound queue whose binding key matches the routing key but as a JMS selector argument has been specified, the expression is evaluated against each matching message. Only messages whose message header values or properties match the expression are routed to the queue.

In the case illustrated, `sub1` has received the yellow and blue message as their property "`area`" cause expression "`area in ('Forties', 'Cromarty')`" to evaluate true. Similarly, the yellow message has also gone to `gale_alert` as its property "`speed`" causes expression "`speed > 7 and speed < 10`" to evaluate true.

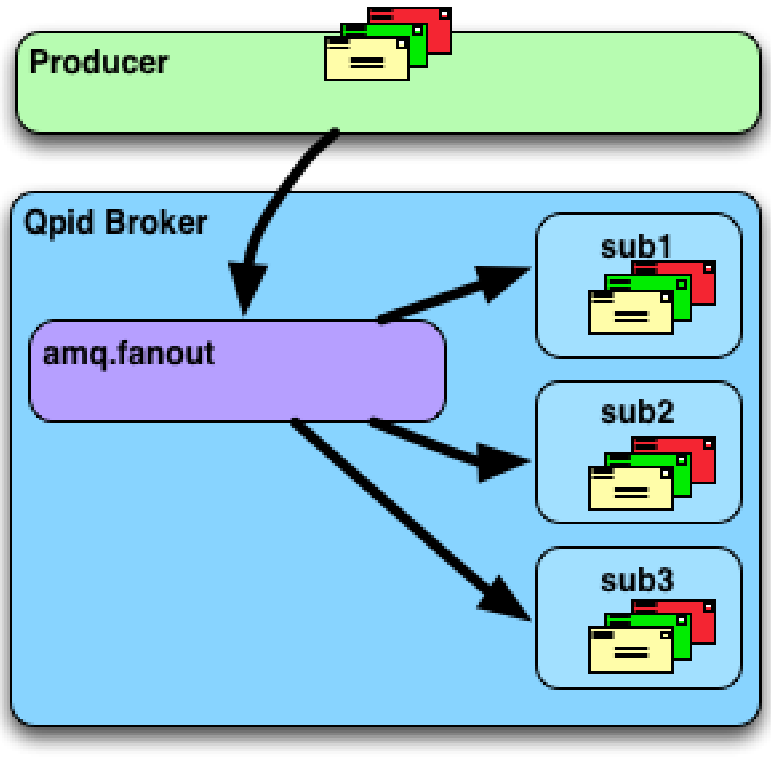
The properties of purple message cause no expressions to evaluate true, so the message is unroutable. It is handled as described in Section 4.3.4, “Unrouteable Messages”.

4.3.2.3. Fanout

The fanout exchange type routes messages to all queues bound to the exchange, regardless of the message's routing key.

Filter rules may be specified using a binding argument specifying a JMS message selector.

Figure 4.7. Fanout exchange



4.3.2.4. Headers

The headers exchange type routes messages to queues based on header properties within the message. The message is passed to a queue if the header properties of the message satisfy the x-match expression specified by the binding arguments with which the queue was bound.

4.3.3. Binding Arguments

Binding arguments are used by certain exchange types to further filter messages.

4.3.3.1. JMS Selector

The binding argument `x-filter-jms-selector` specifies a JMS selector conditional expression. The expression is written in terms of message header and message property names. If the expression evaluates to true, the message is routed to the queue. This type of binding argument is understood by exchange types direct, topic and fanout.[1].

[1] This is a Qpid specific extension.

4.3.3.2. x-match

The binding argument `x-match` is understood by exchange type headers. It can take two values, dictating how the rest of the name value pairs are treated during matching.

- `all` implies that all the other pairs must match the headers property of a message for that message to be routed (i.e. an AND match)
- `any` implies that the message should be routed if any of the fields in the headers property match one of the fields in the arguments table (i.e. an OR match)

A field in the bind arguments matches a field in the message if either the field in the bind arguments has no value and a field of the same name is present in the message headers or if the field in the bind arguments has a value and a field of the same name exists in the message headers and has that same value.

4.3.4. Unrouteable Messages

If an exchange is unable to route a message to any queues, the Broker will:

- If using AMQP 0-10 protocol, and an alternate exchange has been set on the exchange, the message is routed to the alternate exchange. The alternate exchange routes the message according to its routing algorithm and its binding table. If the messages is still unroutable, the message is discarded.
- If using AMQP protocols 0-8..0-9-1, and the publisher set the mandatory flag and the close when no route feature did not close the connection, the message is returned to the Producer.
- Otherwise, the message is discarded.

4.4. Queues

*Queue*s are named entities within a *Virtual Host* that hold/buffer messages for delivery to consumer applications.

Different types of *Queue* have different delivery semantics. The following Queues types are currently supported:

- Standard: a simple First-In-First-Out (FIFO) queue
- Priority: delivery order depends on the priority of each message
- Sorted: delivery order depends on the value of the sorting key property in each message
- Last Value Queue: also known as an LVQ, retains only the last (newest) message received with a given LVQ key value.

Queue configuration details are covered in Chapter 9, *Queues*.

4.5. Ports

The Broker supports configuration of *Ports* to specify the particular AMQP messaging and HTTP/JMX management connectivity it offers for use.

Each Port is configured with the particular *Protocols* and *Transports* it supports, as well as the *Authentication Provider* to be used to authenticate connections. Where SSL is in use, the *Port* configuration also defines which *Keystore* to use and (where supported) which *TrustStore(s)* and whether Client Certificates should be requested/required.

Different *Ports* can support different protocols, and many *Ports* can be configured on the Broker.

The following AMQP protocols are currently supported by the Broker:

- *AMQP 0-8*
- *AMQP 0-9*
- *AMQP 0-9-1*
- *AMQP 0-10*
- *AMQP 1.0*

Addittionally, HTTP and JMX ports can be configured for use by the associated management plugins.

Configuration details for the Ports are covered in Chapter 6, *Broker Ports*.

4.6. Authentication Providers

Authentication Providers are used to authenticate connections to *Ports*. Many *Authentication Providers* can be configured on the Broker at the same time, from which each *Port* can be assigned one.

The following authentication providers are supported:

- Anonymous: allows anonymous connections to the broker
- External: delegates to external mechanisms such as SSL Client Certificate Authentication
- Kerberos: uses Kerberos to authenticate connections via GSS-API.
- SimpleLDAP: authenticate users against an LDAP server.
- PlainPasswordFile: authenticate users against credentials stored in plain text in a local file.
- Base64MD5PasswordFile: authenticate users against credentials stored encoded in a local file.

The Password File based providers can perform explicit management (adding, removing, changing passwords) of users via the Brokers management interfaces. The other providers offer no ability to manage users as they either have no scope for user management (e.g Anonymous) or delegate this task to other systems (e.g LDAP).

The configuration details for Authentication Providers are covered in Section 11.1, “Authentication Providers”.

4.7. Other Services

The Broker can also have *Access Control Providers*, *Group Providers*, *Keystores*, *Trustores* and [Management] *Plugins* configured.

4.7.1. Access Control Providers

Access Control Providers are used to authorize various operations relating to Broker objects.

Access Control Provider configuration and management details are covered in Section 11.3, “Access Control Lists”.

4.7.2. Group Providers

Group Providers are used to aggregate authenticated user principals into groups which can be then be used in Access Control rules applicable to the whole group.

Group Provider configuration and management is covered in Section 11.2, “Group Providers”.

4.7.3. Keystores

Keystores are used to configure details of keystores holding SSL keys and certificates for the SSL transports on Ports.

Keystore configuration and management is covered in Section 11.4.1, “Keystore Configuration”.

4.7.4. Truststores

Truststores are used to configure details of keystores holding SSL certificates for trusting Client Certificate on SSL ports.

Truststore configuration and management is covered in Section 11.4.2, “Truststore / Client Certificate Authentication”.

Chapter 5. Configuring And Managing

5.1. Broker Configuration Store

5.1.1. Introduction

The Broker supports configuration of all its primary components via its HTTP management interface, using the Web Management Console.

The configuration for each component is stored as an entry in the broker configuration store, currently implemented as either a JSON file which persists changes to disk, or an in-memory store which does not. The following components configuration is stored there:

- Broker
- Virtual Host
- Port
- Authentication Provider
- Access Control Provider
- Group Provider
- Key store
- Trust store
- Plugin

Broker startup involves two configuration related items, the 'Initial Configuration' and the Configuration Store. When the broker is started, if a Configuration Store does not exist at the current store location then one will be initialised with the current 'Initial Configuration'. Unless otherwise requested to overwrite the configuration store then subsequent broker restarts will use the existing configuration store and ignore the contents of the 'Initial Configuration'.

5.1.2. Configuration Store Location

The broker will default to using ${qpid.work_dir}/config.json as the path for its configuration store unless otherwise instructed.

The command line argument *-sp* (or *--store-path*) can optionally be used to specify a different relative or absolute path to use for the broker configuration store:

```
$ ./qpid-server -sp ./my-broker-configuration.json
```

If no configuration store exists at the specified/defaulted location when the broker starts then one will be initialised using the current 'Initial Configuration'.

5.1.3. 'Initial Configuration' Location

The 'Initial Configuration' JSON file is used when initialiasing new broker configuration stores. The broker will default to using an internal file within its jar unless otherwise instructed.

The command line argument *-icp* (or *--initial-config-path*) can be used to override the brokers internal file and supply a user-created one:

```
$ ./qpid-server -icp ./my-initial-configuration.json
```

If a Configuration Store already exists at the current store location then the current 'Initial Configuration' will be ignored unless otherwise requested to overwrite the configuration store

5.1.4. Creating an 'Initial Configuration' JSON File

It is possible to have the broker output its default internal 'Initial Configuration' file to disk using the command line argument *-cic* (or *--create-initial-config*). If the option is used without providing a path, a file called *initial-config.json* will be created in the current directory, or alternatively the file can be created at a specified location:

```
$ ./qpid-server -cic ./initial-config.json
```

The 'Initial Configuration' JSON file shares a common format with the brokers JSON Configuration Store implementation, so it is possible to use a brokers Configuration Store output as an initial configuration. Typically 'Initial Configuration' files would not to contain IDs for the configured entities, so that IDs will be generated when the configuration store is initialised and prevent use of the same IDs across multiple brokers, however it may prove useful to include IDs if using the Memory Configuration Store Type.

It can be useful to use Configuration Properties within 'Initial Configuration' files to allow a degree of customisation with an otherwise fixed file.

For an example file, see Section 5.1.8, “Example of JSON 'Initial Configuration'”

5.1.5. Overwriting An Existing Configuration Store

If a configuration store already exists at the configured store location then it is used and the current 'Initial Configuration' is ignored.

The command line argument *-os* (or *--overwrite-store*) can be used to force a new broker configuration store to be initialised from the current 'Initial Configuration' even if one exists:

```
$ ./qpid-server -os -icp ./my-initial-configuration.json
```

This can be useful to effectively play configuration into one or more broker to pre-configure them to a particular state, or alternatively to ensure a broker is always started with a fixed configuration. In the latter case, use of the Memory Configuration Store Type may also be useful.

5.1.6. Configuration Store Type

There are currently two implementations of the pluggable Broker Configuration Store, the default one which persists content to disk in a JSON file, and another which operates only in-memory and so does not retain changes across broker restarts and always relies on the current 'Initial Configuration' to provide the configuration to start the broker with.

The command line argument *-st* (or *--store-type*) can be used to override the default *json*)configuration store type and allow choosing an alterative, such as *memory*)

```
$ ./qpid-server -st memory
```

This can be useful when running tests, or always wishing to start the broker with the same 'Initial Configuration'

5.1.7. Customising Configuration using Configuration Properties

It is posible for 'Initial Configuration' (and Configuration Store) files to contain ${properties} that can be resolved to String values at startup, allowing a degree of customisation using a fixed file. Configuration Property values can be set either via Java System Properties, or by specifying ConfigurationPproperties on the broker command line. If both are defined, System Property values take precedence.

The broker has the following set of core configuration properties, with the indicated default values if not otherwise configured by the user:

Table 5.1. Base Configuration Properties

Name	Description	Value
qpid.amqp_port	Port number used for the brokers default AMQP messaging port	"5672"
qpid.http_port	Port number used for the brokers default HTTP management port	"8080"
qpid.rmi_port	Port number used for the brokers default RMI Registry port, to advertise the JMX ConnectorServer.	"8999"
qpid.jmx_port	Port number used for the brokers default JMX port	"9099"
qpid.home_dir	Location of the broker installation directory, which contains the 'lib' directory and the 'etc' directory often used to store files such as group and ACL files.	Defaults to the value set into the QPID_HOME system property if it is set, or remains unset otherwise unless configured by the user.
qpid.work_dir	Location of the broker working directory, which might contain the persistent message store and broker configuration store files.	Defaults to the value set into the QPID_WORK system property if it is set, or the 'work' subdirectory of the JVMs current working directory.

Use of these core properties can be seen in the default 'Initial Configuration' example.

Configuration Properties can be set on the command line using the *-prop* (or *--configuration-property*) command line argument:

```
$ ./qpid-server -prop "qpid.amqp_port=10000" -prop "qpid.http_port=10001"
```

In the example above, property used to set the port number of the default AMQP port is specified with the value 10000, overriding the default value of 5672, and similarly the vlaue 10001 is used to override the default HTTP port number of 8080. When using the 'Initial Configuration' to initialise a new Configuration Store (either at first broker startup, when requesting to overwrite the configuration store) these new values will be used for the port numbers instead.

NOTE: when saving the broker Configuration Store, either during initialisation when generating any required IDs for the 'Initial Configuration', or when required following user-prompted change via the managmenet interface, values are stored in their resolved state. As such, if a Configuration Store already exists when the broker is started, it is likely that setting a Configuration Property to a value different than it was previously set could have no effect.

NOTE: When running the broker on Windows and starting it via the qpid-server.bat file, the "name=value" argument MUST be quoted.

5.1.8. Example of JSON 'Initial Configuration'

An example of the default 'Initial Configuration' JSON file the broker uses is provided below:

Example 5.1. JSON 'Initial configuration' File

```
{
  "name" : "Broker",
  "defaultVirtualHost" : "default",
  "modelVersion" : "1.0",
  "storeVersion" : 1,
  "authenticationproviders" : [ {
    "name" : "passwordFile",
    "path" : "${qpid.work_dir}/etc/passwd",
    "type" : "PlainPasswordFile"
  } ],
  "ports" : [ {
    "authenticationProvider" : "passwordFile",
    "name" : "HTTP",
    "port" : "8080",
    "protocols" : [ "HTTP" ]
  }, {
    "authenticationProvider" : "passwordFile",
    "name" : "JMX_CONNECTOR",
    "port" : "9099",
    "protocols" : [ "JMX_RMI" ]
  }, {
    "name" : "RMI_REGISTRY",
    "port" : "8999",
    "protocols" : [ "RMI" ]
  }, {
    "name" : "AMQP",
    "port" : "5672"
  } ],
  "virtualhosts" : [ {
    "name" : "default",
    "storePath" : "${qpid.work_dir}/derbystore/default",
    "storeType" : "DERBY"
  } ],
  "plugins" : [ {
    "name" : "jmxManagement",
    "pluginType" : "MANAGEMENT-JMX"
  }, {
    "name" : "httpManagement",
    "pluginType" : "MANAGEMENT-HTTP"
  } ]
}
```

In the configuration above the following entries are stored:

- Authentication Provider of type *PlainPasswordFile* with name "passwordFile"
- Four Port entries: "AMQP", "HTTP", "RMI_REGISTRY", "JMX_CONNECTOR"
- Virtual Host with name "default" and DERBY message store type at location "${qpid.work_dir}/derbystore/default".
- Two management plugins: "jmxManagement" of type "MANAGEMENT-JMX" and "httpManagement" of type "MANAGEMENT-HTTP".
- Broker attributes are stored as a root entry.

5.1.9. Configuring Broker Attributes

The Broker Attributes can be configured using REST Management interfaces and Web Management Console.

The Broker attributes can be changed from Web Management Console by clicking on "Edit" button on "Broker Attributes" panel from Broker tab.

5.2. HTTP Management

5.2.1. Introduction

The brokers HTTP Management Plugin provides the Web Management Console to enable fully configuring the Broker, via an underlying REST management interface.

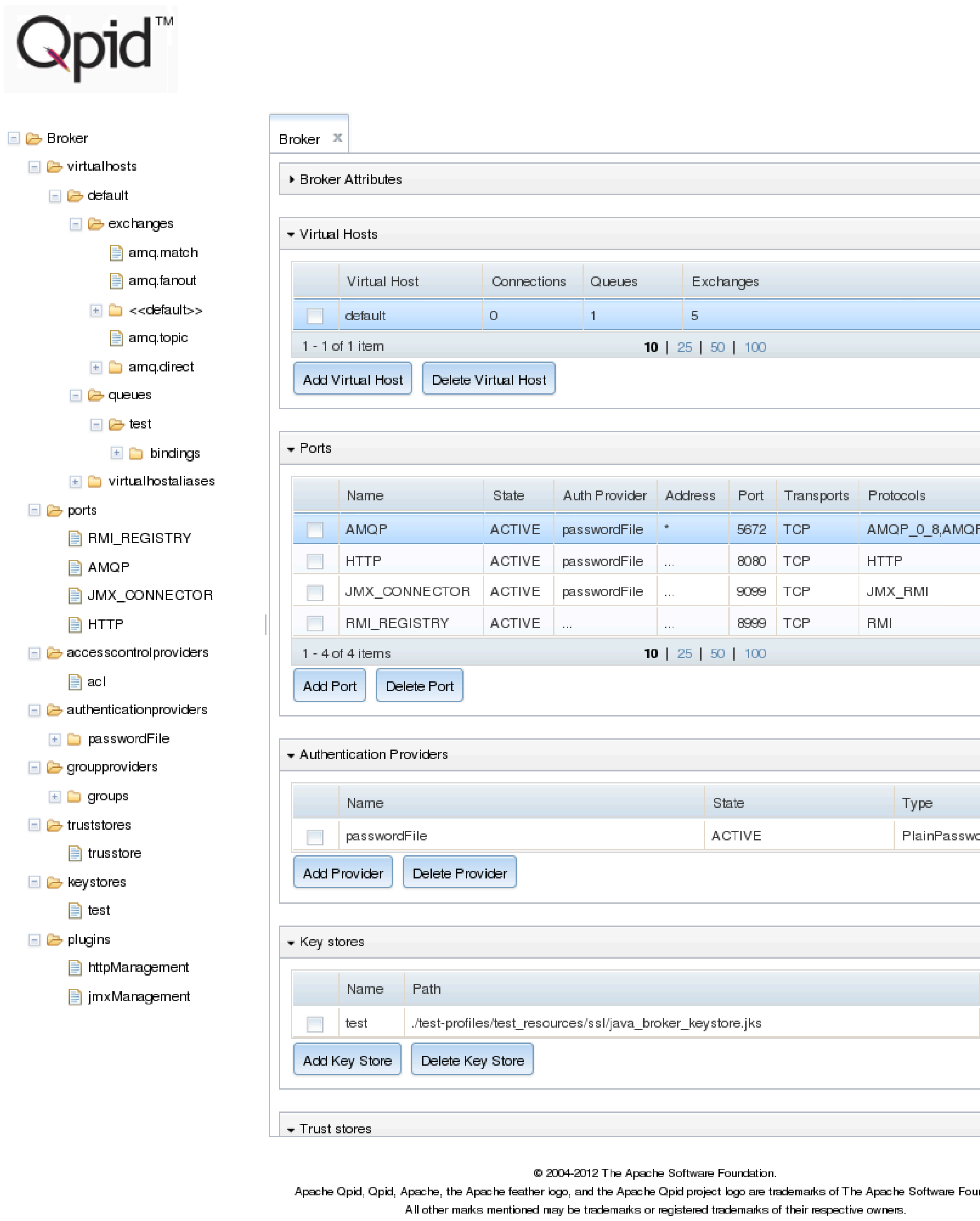
It is included into the brokers Initial Configuration by default, and is responsible for servicing the HTTP ports configured on the broker.

5.2.2. Web Management Console

The Web Management Console can be accessed from a web browser using the URL: http(s)://<hostname>:<port>/management where:

- *hostname* is the broker hostname
- *port* is the HTTP(S) port number

For authenticated and authorized user the page like below should be displayed on navigation to the management URL:



Qpid™

- Broker
  - virtualhosts
    - default
      - exchanges
        - amq.match
        - amq.fanout
        - <<default>>
        - amq.topic
        - amq.direct
      - queues
        - test
          - bindings
      - virtualhostaliases
  - ports
    - RMI_REGISTRY
    - AMQP
    - JMX_CONNECTOR
    - HTTP
  - accesscontrolproviders
    - acl
  - authenticationproviders
    - passwordFile
  - groupproviders
    - groups
  - truststores
    - trusstore
  - keystores
    - test
  - plugins
    - httpManagement
    - jmxManagement

Broker ✕

▸ Broker Attributes

▾ Virtual Hosts

	Virtual Host	Connections	Queues	Exchanges
☐	default	0	1	5

1 - 1 of 1 item **10** \| 25 \| 50 \| 100

Add Virtual Host   Delete Virtual Host

▾ Ports

	Name	State	Auth Provider	Address	Port	Transports	Protocols
☐	AMQP	ACTIVE	passwordFile	*	5672	TCP	AMQP_0_8,AMQF
☐	HTTP	ACTIVE	passwordFile	...	8080	TCP	HTTP
☐	JMX_CONNECTOR	ACTIVE	passwordFile	...	9099	TCP	JMX_RMI
☐	RMI_REGISTRY	ACTIVE	...	...	8999	TCP	RMI

1 - 4 of 4 items **10** \| 25 \| 50 \| 100

Add Port   Delete Port

▾ Authentication Providers

	Name	State	Type
☐	passwordFile	ACTIVE	PlainPasswo

Add Provider   Delete Provider

▾ Key stores

	Name	Path
☐	test	./test-profiles/test_resources/ssl/java_broker_keystore.jks

Add Key Store   Delete Key Store

▾ Trust stores

© 2004-2012 The Apache Software Foundation.
Apache Qpid, Qpid, Apache, the Apache feather logo, and the Apache Qpid project logo are trademarks of The Apache Software Fou
All other marks mentioned may be trademarks or registered trademarks of their respective owners.

[images/Management-Web-Console.png]

5.2.3. HTTP Management Plugin Configuration

The HTTP Management Plugin itself can be configured through the Web Management Console and underlying REST management interface. By double-clicking on the Http Management Plugin name in the object tree a tab for the plugin is displayed with its current settings, which can be changed by clicking on the "Edit" button. The following attributes can be set on the HTTP Management Plugin:

- *Basic Authentication for HTTP*. It is set to false (disabled) by default.
- *Basic Authentication for HTTPS*. It is set to true (enabled) by default.
- *SASL Authentication for HTTP*. It is set to true (enabled) by default.
- *SASL Authentication for HTTPS*. It is set to true (enabled) by default.
- *Session timeout* is the timeout in seconds to close the HTTP session. It is set to 10 minutes by default.

NOTE: Changes to the Session Timeout attribute only take effect at broker restart.

5.2.4. REST API

The brokers Web Management Console makes calls to an underlying REST API to manage the broker. This section provides a brief overview of the REST interface, which can be used directly to monitor and manage the Broker instance although it is still evolving toward being fully considered a seperately supported interface.

The brokers REST interface support traditional REST model which uses the GET method requests to retrieve the information about broker configured objects, DELETE method requests to delete the configured object, PUT to create or update the configured object and POST to perform the configured objects updates not available with the PUT requests.

The table below lists the available REST services with brief description how they can be used.

Table 5.2. Rest services

REST Service URL	Description	GET	PUT	POST	DELETE
/rest/broker	Rest service to manage broker instance	Retrieves the details of broker configuration	Updates broker attributes	Not implemented yet	Not implemented yet
/rest/authenticationprovider /rest/authenticationprovider/<authentication provider name>	Rest service to manage authentication providers on the broker	Retrieves the details about authentication providers	Creates or updates authentication providers	Not implemented yet	Deletes authentication providers
/rest/user /rest/user/<authentication provider name>/<user name>	Rest service to manage user account	Retrieves the details about user account	Creates user account, updates user password	Not implemented yet	Deletes user account

REST Service URL	**Description**	**GET**	**PUT**	**POST**	**DELETE**
/rest/groupprovider /rest/groupprovider/<group provider name>	Rest service to manage group providers	Retrieves the details about group provider(s)	Creates group provider	Not implemented yet	Deletes groups providers
/rest/group /rest/group/<group provider name>/<group name>	Rest service to manage user group	Retrieves the details about user group	Creates group	Not implemented yet	Deletes group
/rest/groupmember /rest/groupmember/<group provider name >/<group name>/<user name>	Rest service to manage group member(s)	Retrieves the details about group member(s)	Add user to group	Not implemented yet	Deletes user from group
/rest/port /rest/port/<port name>	Rest service to manage broker ports(s)	Retrieves the details about the broker port(s)	Creates or updates port	Not implemented yet	Deletes ports
/rest/queue /rest/queue/<virtual host name>/<queue name>	Rest service to manage queue(s)	Retrieves the details about the queue(s)	Creates queue	Not implemented yet	Deletes queue
/rest/exchange /rest/exchange/<virtual host name>/<exchange name>	Rest service to manage exchange(s)	Retrieves the details about the exchange(s)	Creates exchange	Not implemented yet	Deletes exchange
/rest/binding /rest/binding/<virtual host name>/<exchange name>/<queue name>/<binding name>	Rest service to manage binding(s)	Retrieves the details about the binding(s)	Binds a queue to an exchange	Not implemented yet	Deletes binding

REST Service URL	**Description**	**GET**	**PUT**	**POST**	**DELETE**
/rest/connection /rest/connection/<virtual host name>/<connection name>	Rest service to manage connection(s)	Retrieves the details about the connection(s)	Not implemented yet	Not implemented yet	Not implemented yet
/rest/session /rest/session/<virtual host name>/<connection name>/<session name>	Rest service to manage session(s)	Retrieves the details about the session(s)	Not implemented yet	Not implemented yet	Not implemented yet
/rest/message/<virtual host name>/<queue name>	Rest service to manage messages(s)	Retrieves the details about the messages(s)	Not implemented yet	Copies, moves messages	Deletes messages
/rest/message-content/<virtual host name>/<queue name>	Rest service to retrieve message content	Retrieves the message content	Not implemented yet	Not implemented yet	Not implemented yet
/rest/logrecords	Rest service to retrieve broker logs	Retrieves the broker logs	Not implemented yet	Not implemented yet	Not implemented yet
/rest/sasl	Sasl authentication	Retrieves user current authentication status and broker supported SASL mechanisms	Authenticates user using supported SASL mechanisms	Not implemented yet	Not implemented yet
/rest/logout	Log outs	Log outs user	Not implemented yet	Not implemented yet	Not implemented yet
/rest/accesscontrolprovider	Rest service to manage access control providers	Retrieves the details about access control providers	Creates access control provider	Not implemented yet	Deletes access control provider(s)
/rest/keystore	Rest service to manage KeyStores	Retrieves the details about KeyStore	Creates or updates KeyStore	Not implemented yet	Deletes KeyStore(s)
/rest/truststore	Rest service to manage TrustStore	Retrieves the details about TrustStore	Creates or updates TrustStore	Not implemented yet	Deletes TrustStore(s)

REST Service URL	Description	GET	PUT	POST	DELETE
/rest/plugin	Rest service to manage plugins	Retrieves the details about plugins	Updates plugin attributes	Not implemented yet	Not implemented yet

The REST URLs are hierarchical. It is permitted to replace rest URL elements with an "asterisks" in GET requests to denote all object of a particular type. Additionally, trailing object type in the URL hierarchy can be omitted. In this case GET request will return all of the object underneath of the current object.

For example, for binding URL http://localhost:8080/rest/binding/<vhost>/<exchange>/<queue>/<binding> replacing of *<exchange>* with "asterisks" (http://localhost:8080/rest/binding/<vhost>/*/<queue>/<binding>) will result in the GET response containing the list of bindings for all of the exchanges in the virtual host having the given name and given queue. If *<binding>* and *<queue>* are omitted in binding REST URL (http://localhost:8080/rest/binding/<vhostname>/<exchangename>) the GET request will result in returning all bindings for all queues for the given exchange in the virtual host.

Example 5.2. Examples of queue creation using curl (authenticating as user admin):

```
#create a durable queue
curl --user admin -X PUT  -d '{"durable":true}' http://localhost:8080/rest/queue/<
#create a durable priority queue
curl --user admin -X PUT  -d '{"durable":true,"type":"priority"}' http://localhost
```

Example 5.3. Example of binding a queue to an exchange using curl

```
curl --user admin -X PUT  -d '{}' http://localhost:8080/rest/binding/<vhostname>/<
```

NOTE: These curl examples utilise unsecure HTTP transport. To use the examples it is first necessary enable Basic authentication for HTTP within the HTTP Management Configuration (it is off by default). For details see Section 5.2.3, “HTTP Management Plugin Configuration”

5.3. JMX Management

5.3.1. Introduction

The JMX management plugin provides a series of managed beans (MBeans) allowing you to control and monitor the Broker via an industry compliant interface. This provides a convenient intergration point for a variety of Infrastructure Monitoring Solutions, tools such as Jconsole and VisualVM, as well as custom Java programs and scripts.

The following sections describe how to connect to JMX, the configuration of the JMX plugin covering topis including securing with SSL, programmatically interacting with Qpid MBeans and finally a summary of all the MBeans made available from by the plugin.

Important

For new development work, the reader is directed towards the strategic Web Management Console and the REST API. Use the Web/REST interfaces in preference to JMX whenever possible. The JMX interface may be withdrawn in a future release.

5.3.2. Default Configuration

By default, the Broker is shipped with JMX enabled.

The RMI registry port runs on port `8999` and the JMX connector on port `9099`. The connector is not SSL protected. Qpid will use the Platform MBeanServer [http://docs.oracle.com/javase/6/docs/api/java/lang/management/ManagementFactory.html#getPlatformMBeanServer()].

To change these settings, use the Web Management interface.

5.3.3. Connecting to JMX

The following example uses Jconsole to illustrates how to connect to JMX and assume the defaults described above. Jconsole is a management tool that comes with the JDK. It provides a very simple view of the MBeans, but requires no special configuration to be used with Qpid.

For full details of Jconsole itself refer to Oracle's JConsole Guide [http://java.sun.com/javase/6/docs/technotes/guides/management/jconsole.html].

Jconsole can be used to connect to local or remote Java processes. On startup, it presents a list of all the Java processes running on the local host and also allows you to specify a service url to connect to a Java process running on a remote host.

To start Jconsole on the command line, type:

```
jconsole
```

5.3.3.1. Local

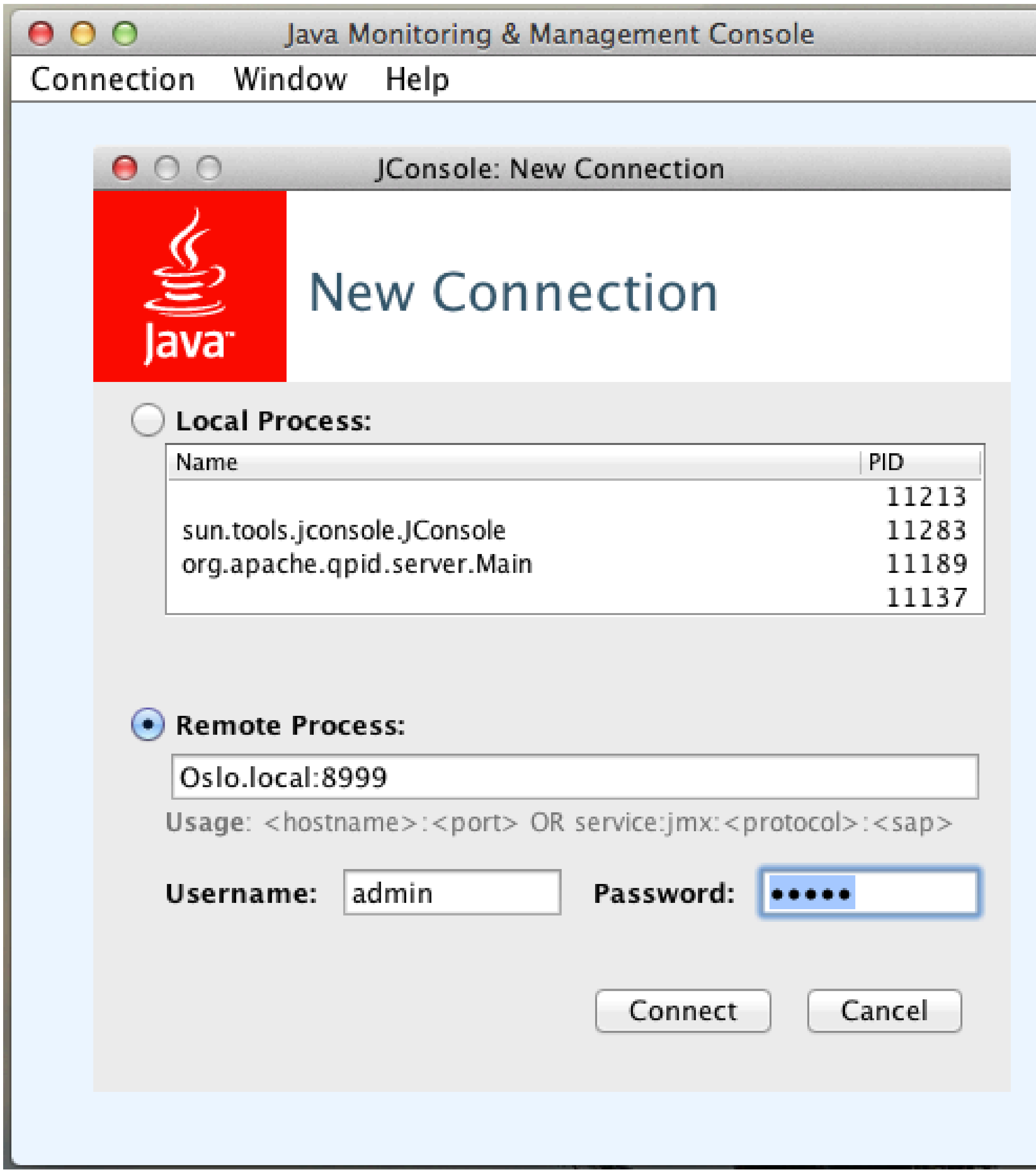
To connect to a Broker running locally, simply select the process from the list. You can identify the Broker by looking for its classname `org.apache.qpid.server.Main`.

5.3.3.2. Remote

To connect to a broker running remotely, provide the hostname and port number of the *RMI registry port* (e.g. `hostname:8999`) and a valid username and password.

You can also provide a service url in the form `service:jmx:rmi:///jndi/rmi://hostname:8999/jmxrmi`

Figure 5.1. Making a remote JMX connection to a Broker using jconsole



Local Process:

Name	PID
	11213
sun.tools.jconsole.JConsole	11283
org.apache.qpid.server.Main	11189
	11137

[✓ Remote Process:]

Oslo.local:8999

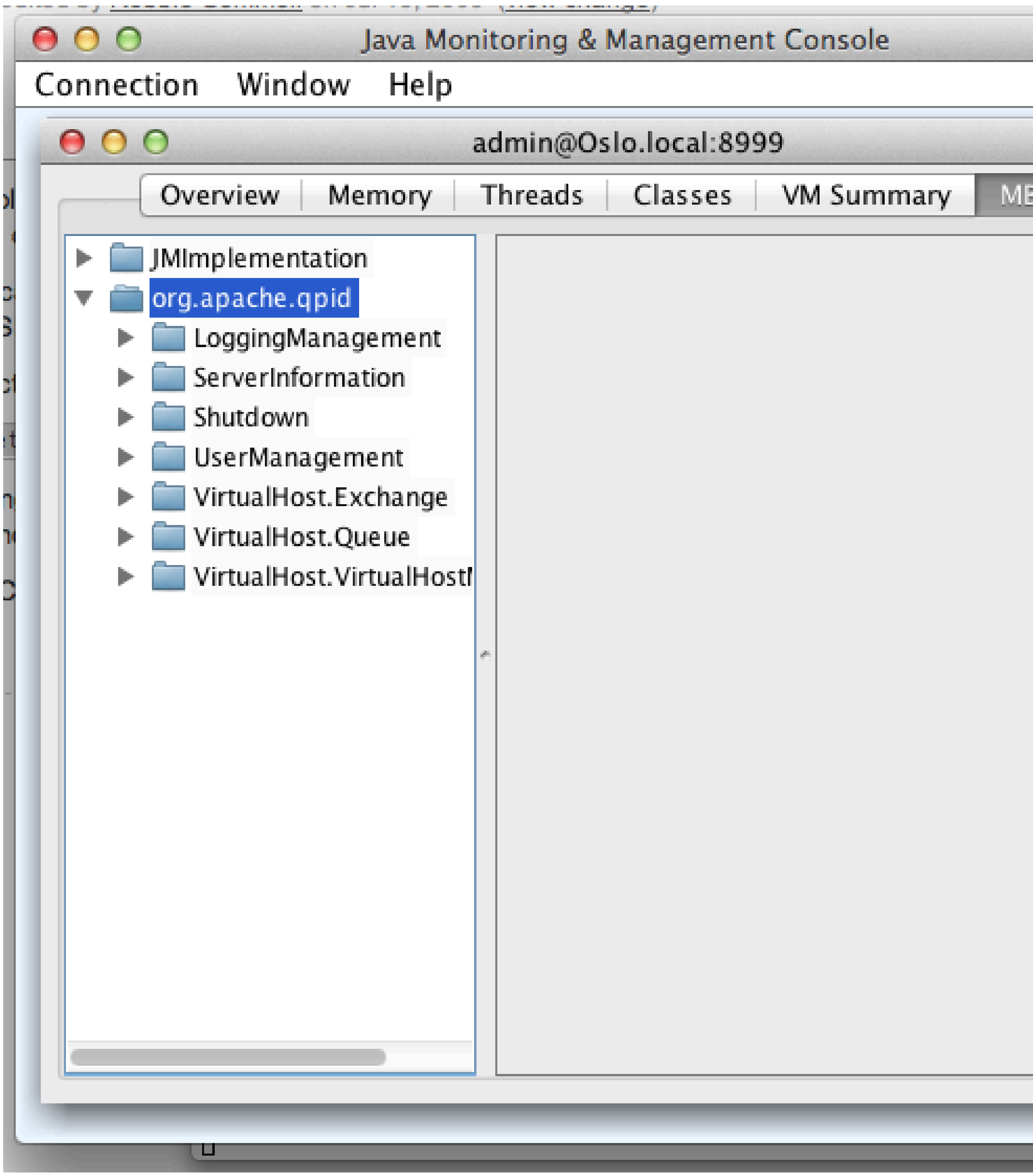
Usage: <hostname>:<port> OR service:jmx:<protocol>:<sap>

Username: admin

Password: •••••

Once you are connected expand the tree of nodes marked `org.apache.qpid` to begin to interact with the Qpid MBeans.

Figure 5.2. Qpid MBean hierarchy



5.3.3.3. Connecting to a remote Broker protected by SSL

If you are connecting to a remote Broker whose JMX connector port has been secured with SSL certificate signed by a private CA (or a self-signed certificate), you will need to pass a trust store and trust store password to Jconsole. If this is required, start jconsole with the following options:

```
jconsole -J-Djavax.net.ssl.trustStore=jmxtruststore.jks -J-Djavax.net.ssl.trustSto
```

5.3.4. Example JMX Client

The following java snippet illustrates a JMX client that connects to Qpid over JMX passing a userid and password, looks up the ManagedBroker [http://svn.apache.org/viewvc/qpid/trunk/qpid/java/management/common/src/main/java/org/apache/qpid/management/common/mbeans/ManagedBroker.java?view=co] object corresponding to the `myvhost` virtualhost, then invokes a method on the virtualhost to create a new queue.

A full introduction to custom JMX clients is beyond the scope of this book. For this the reader is directed toward Oracle's JMX tutorial. [http://docs.oracle.com/javase/tutorial/jmx/]

Example 5.4. JMX Client illustrating the creation of a new queue

```
Map<String, Object< environment = new HashMap<String, Object>();
environment.put(JMXConnector.CREDENTIALS, new String[] {"admin","password"});
// Connect to service
JMXServiceURL url =  new JMXServiceURL("service:jmx:rmi:///jndi/rmi://localhost:89
JMXConnector jmxConnector = JMXConnectorFactory.connect(url, environment);
MBeanServerConnection mbsc =  jmxConnector.getMBeanServerConnection();
// Object name for ManagedBroker mbean for virtualhost myvhost
ObjectName objectName = new ObjectName("org.apache.qpid:type=VirtualHost.VirtualHo
// Get the ManagedBroker object
ManagedBroker managedBroker = JMX.newMBeanProxy(mbsc, objectName, ManagedBroker.cl

// Create the queue named "myqueue"
managedBroker.createNewQueue("myqueue", null, true);
```

The Qpid classes required for a custom JMX client are included in the `qpid-management-common` artefact.

5.3.5. The MBeans

The following table summarises the available MBeans. The MBeans are self-describing: each attribute and operation carry a description describing their purpose. This description is visible when using tools such Jconsole. They are also available on Management interfaces themselves (linked below).

Table 5.3. Qpid Broker MBeans

Management Interface	Object Name
ManagedBroker [http://svn.apache.org/viewvc/qpid/trunk/qpid/java/management/common/src/main/java/org/apache/qpid/management/common/mbeans/ManagedBroker.java?view=co]	`org.apache.qpid:type=VirtualHost.VirtualHostMan`
	MBean corresponding to the named virtualhost. Allows operations such as the creation/deletion of queues and exchanges on that virtualhost and virtualhost levell statistics.

Management Interface	**Object Name**
ManagedQueue [http://svn.apache.org/viewvc/qpid/trunk/qpid/java/management/common/src/main/java/org/apache/qpid/management/common/mbeans/ManagedQueue.java?view=co]	`org.apache.qpid:type=VirtualHost.Queue,VirtualH`
	MBean corresponding to the named queue on the given virtualhost. Allows queue management operations such as view message, move message and clear queue. Exposes attributes such as queue depth and durability.
ManagedExchange [http://svn.apache.org/viewvc/qpid/trunk/qpid/java/management/common/src/main/java/org/apache/qpid/management/common/mbeans/ManagedExchange.java?view=co]	`org.apache.qpid:type=VirtualHost.Exchange,Virtu`
	MBean corresponding to the named exchange on the given virtualhost. Allows exchange management operations such as the creation and removal of bindings. The supported exchange types are exposed by the `exchangeTypes` attribute of the virtualhost.
ManagedConnection [http://svn.apache.org/viewvc/qpid/trunk/qpid/java/management/common/src/main/java/org/apache/qpid/management/common/mbeans/ManagedConnection.java?view=co]	`org.apache.qpid:type=VirtualHost.Connection,Vir` *`peerid:ephemeralport`*`"`
	MBean representing a active AMQP connection to the named virtual host. Name is formed from the IP and ephemeral port of the peer. Attributes include the client version and connection level statistics.
UserManagement [http://svn.apache.org/viewvc/qpid/trunk/qpid/java/management/common/src/main/java/org/apache/qpid/management/common/mbeans/UserManagement.java?view=co]	`org.apache.qpid:type=UserManagement,name="UserM` *`authentication manager name`*`"`
	When using Plain password provider or Base 64 MD5 password provider, permits user operations such creation and deletion of users. and password changes.
ServerInformation [http://svn.apache.org/viewvc/qpid/trunk/qpid/java/management/common/src/main/java/org/apache/qpid/management/common/mbeans/ServerInformation.java?view=co]	`org.apache.qpid:type=ServerInformation,name=Ser`
	Exposes broker wide statistics, product version number and JMX management API version number.
LoggingManagement [http://svn.apache.org/viewvc/qpid/trunk/qpid/java/management/common/src/main/java/org/apache/qpid/management/common/mbeans/LoggingManagement.java?view=co]	`org.apache.qpid:type=LoggingManagement,name=Log`
	MBean permitting control of the Broker's logging. Exposes operations allow the logging level to be controlled at runtime (without restarting the Broker) and others that allow changes to be written back to the log4j.xml logging configuration file, or the contents of the log4.xml file to be re-read at runtime.

5.4. Other Tooling

Chapter 6. Broker Ports

This section guides through the process of configuring of Broker AMQP and non-AMQP ports.

6.1. Configuring Broker Ports

The Broker Ports can be configured using the HTTP management interfaces.

The following Port managing operations are available from the Web Management Console:

- A new Port can be created by clicking "Add Port" button on the Broker tab.
- An existing Port details are displayed on the Port tab after clicking on Port name in the Broker object tree or after clicking on a Port row in the Ports grid on the Broker tab.
- An existing Port can be edited by clicking on "Edit" button on the Port tab.
- An existing Port can be deleted by clicking on "Delete Port" button on Broker tab or "Delete" button on the Port tab.

Three different types of ports can be created:

- AMQP ports accepting connections for supported AMQP protocols.
- HTTP ports accepting connections for HTTP and HTTPS (by selecting the SSL transport) and used by web management plugin.
- JMX related ports supporting RMI and JMX_RMI protocols and used by JMX management plugin.

It is possible to create any number of HTTP and AMQP (supporting any mixture of AMQP versions) ports, however only two JMX-related ports can recommended to configure on the Broker: one with the RMI protocol for the RMI Registry to advertise the JMX Connector Server and another with the JMX_RMI protocol for the JMX Connector Server itself.

A configured Authentication Provider must be selected on ports using the AMQP, HTTP and JMX_RMI protocols.

SSL can be enabled for Ports with protocols that support it by selecting the 'SSL' transport, at which point a configured KeyStore must also be selected for the Port.

Client Certificate Authentication can be configured for AMQP ports. This requires selecting one or more configured TrustStores on the Port and setting the *needClientAuthentication* and *wantClientAuthentication* attributes as desired. They allow control of whether the client must present an SSL certificate, allowing for three possible states: required (needClientAuth = true), requested (wantClientAuth = true), or none desired (both false, the default). If both elements are set to true, needClientAuth takes precedence. When using Client Certificate Authentication it may be desirable to use the External Authentication Provider.

Important

Changes to port attributes will take effect only after broker restart. You should restart the broker immediately if you require the attribute change sto take effect.

Important

Following deletion of an active HTTP or JMX Port, the port remains bound until the Broker is restarted. You should restart the broker immediately if you require preventing new connections on the port or disconnecting existing clients.

Chapter 7. Virtual Hosts

7.1. Configuring And Managing

One or more Virtual Hosts can be configured on the Broker. The HTTP management interfaces can be used to add and delete Virtual Hosts.

A new Virtual Host can be created in two ways:

- *Supplying simply a store type and a store path*: In this case, the virtual host attributes are currently derived from default attribute values defined on the broker. This is the preferred approach.
- *Supplying the path to a Virtual Host XML configuration file*: In this case, specific per-virtualhost attribute configuration can be set in the file, as well as pre-configuring queues, exchanges, etc. This is no longer the preferred approach and will likely be removed in a future release, however it is currently still neccessary to support certain use-cases such as per-virtualhost attribute configuration, and specialised store configuration such as for the BDB HA Message Store.

The following Virtual Host Managing operations are available from Web Management Console:

- A new Virtual Host can be added into Broker by pressing "Add Virtual Host" button on the Broker tab.
- The existing Virtual Host(s) can be removed by pressing "Remove Virtual Host" button on the Broker tab.
- The Virtual Host details can be viewed on the Virtual Host tab. This tab can be displayed after clicking onto Virtual Host Name in the Broker object tree or onto the Virtual Host row in the Virtual Hosts grid on the Broker tab.
- Queues can be configured (added/removed) from Virtual Host tab
- Exchange can be configured (added/removed) from Virtual Host tab
- Existing Exchange/Queue tabs can be navigated from Virtual Host tab

Chapter 8. Exchanges

8.1. Configuring Virtual Host Exchanges

The Virtual Host Exchanges can be configured using REST Management interfaces, Web Management Console and Virtual Host configuration file.

The following Exchange managing operations are available from Web Management Console:

- A new Exchange can be added by clicking on "Add Exchange" on the Virtual Host tab.
- An existing Exchange details can be viewed the Exchange tab. Exchange tab is shown after clicking on Exchange name in Broker object tree or by clicking on Exchange row in Exchanges grid on Virtual Host tab.
- An existing Exchange can be deleted by clicking on "Delete Exchange" button on Virtual Host tab or "Delete Exchange" button on the Exchange tab.
- An existing Queue can be bound to the Exchange by clicking on "Add Binding" button on the Exchange tab.
- An existing Queue binding can be deleted from Exchange by clicking on "Delete Binding" button on the Exchange tab.

An example of configuring Exchanges in Virtual Host configuration file is provided in Section 14.8, “Configuring Exchanges”.

Chapter 9. Queues

9.1. Queue Types

9.1.1. Introduction

In addition to the standard queue type where messages are delivered in the same order that they were sent, the Java Broker supports three additional queue types which allows for alternative delivery behaviours. These are priority-queues, sorted-queues-, last-value-queues (LVQs). Additionally, Java Broker supports message grouping.

In the following sections, the semantics of each queue type is described, followed by a description of how instances of these queue can be created via configuration, programmatically or Web Management Console.

The final section discusses the importance of using a low client pre-fetch with these queued.

9.1.2. Priority Queues

In a priority queue, messages on the queue are delivered in an order determined by the JMS priority message header [http://docs.oracle.com/javaee/6/api/javax/jms/Message.html#getJMSPriority()] within the message. By default Qpid supports the 10 priority levels mandated by JMS, with priority value 0 as the lowest priority and 9 as the highest.

It is possible to reduce the effective number of priorities if desired.

JMS defines the default message priority [http://docs.oracle.com/javaee/6/api/javax/jms/Message.html#DEFAULT_PRIORITY] as 4. Messages sent without a specified priority use this default.

9.1.3. Sorted Queues

Sorted queues allow the message delivery order to be determined by value of an arbitrary JMS message property [http://docs.oracle.com/javaee/6/api/javax/jms/Message.html#getStringProperty()]. Sort order is alpha-numeric and the property value must have a type java.lang.String.

Messages sent to a sorted queue without the specified JMS message property will be inserted into the 'last' position in the queue.

9.1.4. Last Value Queues (LVQ)

LVQs (or conflation queues) are special queues that automatically discard any message when a newer message arrives with the same key value. The key is specified by arbitrary JMS message property [http://docs.oracle.com/javaee/6/api/javax/jms/Message.html#getPropertyNames()].

An example of an LVQ might be where a queue represents prices on a stock exchange: when you first consume from the queue you get the latest quote for each stock, and then as new prices come in you are sent only these updates.

Like other queues, LVQs can either be browsed or consumed from. When browsing an individual subscriber does not remove the message from the queue when receiving it. This allows for many subscriptions to browse the same LVQ (i.e. you do not need to create and bind a separate LVQ for each subscriber who wishes to receive the contents of the LVQ).

Messages sent to an LVQ without the specified property will be delivered as normal and will never be "replaced".

9.1.5. Creating a Priority, Sorted or LVQ Queue

To create a priority, sorted or LVQ queue, it can be defined in the virtualhost configuration file, can be created programmtically from a client via AMQP (using an extension to JMS), using JMX, using REST interfaces or created in Web Management Console. These methods are described below.

Once a queue is created you cannot change its type (without deleting it and re-creating). Also note you cannot currently mix the natures of these queue types, for instance, you cannot define a queue which it both an LVQ and a priority-queue.

9.1.5.1. Using Web Management Console

On clicking on "Add Queue" button on Virtual Host tab the pop-up dialog to create a queue is displayed.

For a Simple queue a Queue Type "Standard" should be selected

For a Priority queue a Queue Type "Priority" and the priority value (10 by default) should be selected.

For a Sorted queue a Queue Type "Sorted" and Sort Message Property should be specified.

For a LVQ queue a Queue Type "LVQ" and LVQ Message Property should be specified.

Additionally, for each type of the queue Flow Control Thresholds and Alert Thresholds can be specified in optional fields.

Also, a Dead Letter Queue can be configured for the Queue by checking "Create DLQ" check box. The maximum number of delivery retries before message is sent to the DLQ can be specified in "Maximum Delivery Retries" field. However, configuring of maximum delivery retries on a queue without DLQ(AlternateExchange) will result in messages being discarded after the limit is reached.

9.1.5.2. Using JMX or AMQP

To create a priority, sorted or LVQ queue programmatically from JMX or using a Qpid extension to JMS, pass the appropriate queue-declare arguments.

Table 9.1. Queue-declare arguments understood for priority, sorted and LVQ queues

Queue type	Argument name	Argument name	Argument Description
priority	x-qpid-priorities	java.lang.Integer	Specifies a priority queue with given number priorities
sorted	qpid.queue_sort_key	java.lang.String	Specifies sorted queue with given message property used to sort the entries
lvq	qpid.last_value_queue_key	java.lang.String	Specifies lvq queue with given message property used to conflate the entries

The following example illustrates the creation of the a LVQ queue from a javax.jms.Session object. Note that this utilises a Qpid specific extension to JMS and involves casting the session object back to its Qpid base-class.

Example 9.1. Creation of an LVQ using the Qpid extension to JMS

```
Map<String,Object> arguments = new HashMap<String, Object>();
arguments.put("qpid.last_value_queue_key","ISIN");
AMQDestination amqQueue = (AMQDestination) context.lookup("myqueue");
((AMQSession<?,?>) session).createQueue(
        AMQShortString.valueOf(amqQueue.getQueueName()),
        amqQueue.isAutoDelete(),
        amqQueue.isDurable(),
        amqQueue.isExclusive(),
        arguments);
```

The following example illustrates the creation of the sorted queue from a the JMX interface using the ManagedBroker interface.

Example 9.2. Creation of a sorted queue using JMX

```
Map<String, Object> environment = new HashMap<String, Object>();
environment.put(JMXConnector.CREDENTIALS, new String[] {"admin","password"});
// Connect to service
JMXServiceURL url =  new JMXServiceURL("service:jmx:rmi:///jndi/rmi://localhost:89
JMXConnector jmxConnector = JMXConnectorFactory.connect(url, environment);
MBeanServerConnection mbsc =  jmxConnector.getMBeanServerConnection();
// Object name for ManagedBroker for virtualhost myvhost
ObjectName objectName = new ObjectName("org.apache.qpid:type=VirtualHost.VirtualHo
// Get the ManagedBroker object
ManagedBroker managedBroker = JMX.newMBeanProxy(mbsc, objectName, ManagedBroker.cl

// Create the queue passing arguments
Map<String,Object> arguments = new HashMap<String, Object>();
arguments.put("qpid.queue_sort_key","myheader");
managedBroker.createNewQueue("myqueue", null, true, arguments);
```

9.1.5.3. Using configuration

How to declare queues in the Virtual Host configuration file is described in Section 14.9, “Configuring Queues”.

9.1.6. Binding queues to exchanges

Queues can be bound to the broker exchanges in the virtualhost configuration file or programmtically from a client using AMQP bind API (using an extension to JMS), using JMX API, using REST interfaces or Web Management Console.

A queue can be bound to different exchanges at the same time. Also, a queue can be bound to the same exchange multiple times. Differenent binding keys can be used to bind a queue to the same topic or direct exchanges.

Binding attributes can be specified on binding creation to allow filtering of messages accepted by the queue using a selector expression or/and specifying whether messages published by its own connection should be delivered to it.

9.1.6.1. Using Web Management Console

A queue can be bound to an exchange by clicking on "Add Binding" button on a Queue tab or an Exchange tab.

9.1.6.2. Using JMX or AMQP

The following example illustrates the creation of queue binding to topic exchange with JMS client.

Example 9.3. Binding a queue using JMS

```
ConnectionFactory connectionFactory = ...
Connection connection = connectionFactory.createConnection();
AMQSession<?, ?> session = (AMQSession<?,?>)connection.createSession(false, Sessio

...

AMQShortString queueName = new AMQShortString("testQueue");
AMQShortString routingKey = new AMQShortString("testRoutingKey");
AMQDestination destination = (AMQDestination) session.createQueue(queueName.asStri

...

// binding arguments
Map<String, Object> arguments = new HashMap<String, Object>();
arguments.put("x-filter-jms-selector", "application='app1'");

// create binding
session.bindQueue(queueName, routingKey, FieldTable.convertToFieldTable(arguments)
    new AMQShortString("amq.topic"), destination);
```

The following example illustrates the creation of queue binding to topic exchange with JMX interface using the ManagedExchange interface.

Example 9.4. Binding a queue using JMX

```
Map<String, Object> environment = new HashMap<String, Object>();
environment.put(JMXConnector.CREDENTIALS, new String[] {"admin","password"});

// Connect to service
JMXServiceURL url =  new JMXServiceURL("service:jmx:rmi:///jndi/rmi://localhost:89
JMXConnector jmxConnector = JMXConnectorFactory.connect(url, environment);
MBeanServerConnection mbsc =  jmxConnector.getMBeanServerConnection();

// Object name for topic Exchange MBean for virtualhost 'default'
ObjectName objectName = new ObjectName("org.apache.qpid:type=VirtualHost.Exchange,
    + "VirtualHost=\"default\",name=\"amq.topic\",ExchangeType=topic");

// Get the ManagedExchange object
ManagedExchange topicExchange = JMX.newMBeanProxy(mbsc, objectName, ManagedExchang

// Create the binding arguments
Map<String,Object> arguments = new HashMap<String, Object>();
arguments.put("x-filter-jms-selector", "application='app1'");
```

```
// create binding
topicExchange.createNewBinding("queue", "testBindingKey", arguments);
```

9.1.6.3. Using configuration

How to bind queues in the Virtual Host configuration file is shown in Section 14.10, “Queue Binding”.

9.1.7. Messaging Grouping

The broker allows messaging applications to classify a set of related messages as belonging to a group. This allows a message producer to indicate to the consumer that a group of messages should be considered a single logical operation with respect to the application.

The broker can use this group identification to enforce policies controlling how messages from a given group can be distributed to consumers. For instance, the broker can be configured to guarantee all the messages from a particular group are processed in order across multiple consumers.

For example, assume we have a shopping application that manages items in a virtual shopping cart. A user may add an item to their shopping cart, then change their mind and remove it. If the application sends an *add* message to the broker, immediately followed by a *remove* message, they will be queued in the proper order - *add*, followed by *remove*.

However, if there are multiple consumers, it is possible that once a consumer acquires the *add* message, a different consumer may acquire the *remove* message. This allows both messages to be processed in parallel, which could result in a "race" where the *remove* operation is incorrectly performed before the *add* operation.

9.1.7.1. Grouping Messages

In order to group messages, the application would designate a particular message header as containing a message's *group identifier*. The group identifier stored in that header field would be a string value set by the message producer. Messages from the same group would have the same group identifier value. The key that identifies the header must also be known to the message consumers. This allows the consumers to determine a message's assigned group.

The header that is used to hold the group identifier, as well as the values used as group identifiers, are totally under control of the application.

9.1.7.2. The Role of the Broker in Message Grouping

The broker will apply the following processing on each grouped message:

- Enqueue a received message on the destination queue.
- Determine the message's group by examining the message's group identifier header.
- Enforce *consumption ordering* among messages belonging to the same group. *Consumption ordering* means one of two things depending on how the queue has been configured.
  - In default mode, a group gets assigned to a single consumer for the lifetime of that consumer, and the broker will pass all subsequent messages in the group to that consumer.
  - In 'shared groups' mode (which gives the same behaviour as the Qpid C++ Broker) the broker enforces a looser guarantee, namely that all the *currently unacknowledged messages* in a group are sent to the

same consumer, but the consumer used may change over time even if the consumers do not. This means that only one consumer can be processing messages from a particular group at any given time, however if the consumer acknowledges all of its acquired messages then the broker *may* pass the next pending message in that group to a different consumer.

The absence of a value in the designated group header field of a message is treated as follows:

- In default mode, failure for a message to specify a group is treated as a desire for the message not to be grouped at all. Such messages will be distributed to any available consumer, without the ordering quarantees imposed by grouping.
- In 'shared groups' mode (which gives the same behaviour as the Qpid C++ Broker) the broker assigns messages without a group value to a 'default group'. Therefore, all such "unidentified" messages are considered by the broker as part of the same group, which will handled like any other group. The name of this default group is "qpid.no-group", although it can be customised as detailed below.

Note that message grouping has no effect on queue browsers.

Note well that distinct message groups would not block each other from delivery. For example, assume a queue contains messages from two different message groups - say group "A" and group "B" - and they are enqueued such that "A"'s messages are in front of "B". If the first message of group "A" is in the process of being consumed by a client, then the remaining "A" messages are blocked, but the messages of the "B" group are available for consumption by other consumers - even though it is "behind" group "A" in the queue.

9.1.7.3. Broker Message Grouping Configuration

In order for the broker to determine a message's group, the key for the header that contains the group identifier must be provided to the broker via configuration. This is done on a per-queue basis, when the queue is first configured.

This means that message group classification is determined by the message's destination queue.

Specifically, the queue "holds" the header key that is used to find the message's group identifier. All messages arriving at the queue are expected to use the same header key for holding the identifer. Once the message is enqueued, the broker looks up the group identifier in the message's header, and classifies the message by its group.

Message group support is specified by providing one or more of the following settings in the arguments map that is used when declaring the queue (e.g. when calling `AMQSession.createQueue()`).

Table 9.2. Queue Declare Message Group Configuration Arguments

Key	Value
qpid.group_header_key	The name of the message header that holds the group identifier value. The values in this header may be of any supported type (i.e. not just strings).
qpid.shared_msg_group	Provide a value of "1" to switch on 'shared groups' mode.
qpid.default_msg_group	The value to use as the default group when operating in 'shared groups' mode.

The default group for groups operating in 'shared groups' mode can be updated broker-wide using a system property as follows, however do note that the queue declaration argument detailed above takes precedence:

-Dqpid.broker_default-shared-message-group="your.default.shared.group"

It is important to note that there is no need to provide the actual group identifer values that will be used. The broker learns these values as messages are received. Also, there is no practical limit - aside from resource limitations - to the number of different groups that the broker can track at run time.

9.1.8. Using low pre-fetch with special queue types

Qpid clients receive buffered messages in batches, sized according to the pre-fetch value. The current default is 500.

However, if you use the default value you will probably *not* see desirable behaviour when using priority, sorted, lvq or grouped queues. Once the broker has sent a message to the client its delivery order is then fixed, regardless of the special behaviour of the queue.

For example, if using a priority queue and a prefetch of 100, and 100 messages arrive with priority 2, the broker will send these messages to the client. If then a new message arrives will priority 1, the broker cannot leap frog messages of lower priority. The priority 1 will be delivered at the front of the next batch of messages to be sent to the client.

So, you need to set the prefetch values for your client (consumer) to make this sensible. To do this set the Java system property `max_prefetch` on the client environment (using -D) before creating your consumer.

A default for all client connections can be set via a system property:

```
-Dmax_prefetch=1
```

The prefetch can be also be adjusted on a per connection basis by adding a `maxprefetch` value to the Connection URLs [../../Programming-In-Apache-Qpid/html/QpidJNDI.html#section-jms-connection-url]

```
amqp://guest:guest@client1/development?maxprefetch='1'&brokerlist='tcp://localhost
```

Setting the Qpid pre-fetch to 1 will give exact queue-type semantics as perceived by the client however, this brings a performance cost. You could test with a slightly higher pre-fetch to trade-off between throughput and exact semantics.

Chapter 10. Virtual Host Message Stores

10.1. Memory Message Store

The Java broker has an in-memory message store implementation. This section will detail configuration for using the MemoryMessageStore.

Note: when using this store, the broker will store both persistent and non-persistent messages in memory, which is to say that neither will be available following a broker restart, and the ability to store new messages will be entirely constrained by the JVM heap size.

The MemoryMessageStore can be selected on Virtual Host creation via REST Management interfaces and Web Management Console. For details, see Chapter 7, *Virtual Hosts*.

Alternatively, the MemoryMessageStore can configured in Virtual Host configuration xml. For details, see Section 14.3, “Configuring MemoryMessageStore”.

10.2. Derby Message Store

The Java broker has a message store implementation backed by Apache Derby. This section will detail configuration for using the DerbyMessageStore.

The DerbyMessageStore can be selected on Virtual Host creation via REST Management interfaces and Web Management Console. For details, see Chapter 7, *Virtual Hosts*.

Alternatively, the DerbyMessageStore can configured in Virtual Host configuration xml. For details, see Section 14.6, “Configuring DerbyMessageStore”.

10.3. SQL Message Store

The Java broker has a message store implementation backed by JDBC API. This section will detail configuration for using the JDBCMessageStore.

The JDBCMessageStore can be selected on Virtual Host creation via REST Management interfaces and Web Management Console. For details, see Chapter 7, *Virtual Hosts*.

Alternatively, the JDBCMessageStore can configured in Virtual Host configuration xml. For details, see Section 14.7, “Configuring JDBCMessageStore”.

10.3.1. JDBC driver

Only JDBC 4.0 compatible drivers can be used with JDBCMessageStore as it does not register a driver class explicitly. In order to use a JDBCMessageStore a driver library is required to be present in the Broker classpath. For the standard Broker distribution a driver library can be put into ${QPID_HOME}/lib/opt folder.

10.4. BDB Message Store

The Java broker has an *optional* message store implementation backed by Oracle BDB JE. This section will detail where to download the optional dependency from, how to add it to the broker installation, and provide an example configuration for using the BDBMessageStore.

The BDBMessageStore can be selected on Virtual Host creation via REST Management interfaces and Web Management Console. For details, see Chapter 7, *Virtual Hosts*.

Alternatively, the BDBMessageStore can configured in Virtual Host configuration xml. For details, see Section 14.4, “Configuring BDBMessageStore”.

10.4.1. Oracle BDB JE download

The BDB based message store is optional due to its dependency on Oracle BDB JE, which is distributed under the Sleepycat licence. As a result of this, the dependency cant be distributed by the Apache Qpid project as part of the broker release package.

If you wish to use the BDBMessageStore, then you must download the Oracle BDB JE 5.0.97 release from the Oracle website. [http://www.oracle.com/technetwork/products/berkeleydb/downloads/index.html?ssSourceSiteId=ocomen]

The download has a name in the form je-5.0.97.tar.gz. It is recommended that you confirm the integrity of the download by verifying the MD5.

10.4.2. Oracle BDB JE jar installation

If you wish to use the BDBMessageStore, copy the je-5.0.97.jar from within the release downloaded above into the 'opt' sub-directory of the brokers 'lib' directory.

```
Unix:
cp je-5.0.97.jar qpid-broker-0.28/lib/opt

Windows:
copy je-5.0.97.jar qpid-broker-0.28\lib\opt
```

10.5. High Availability BDB Message Store

The Java broker has an *optional* High Availability message store implementation backed by Oracle BDB JE HA. This section references information on where to download the optional dependency from, how to add it to the broker installation, and how to configure the BDBHAMessageStore.

For more detailed information about use of this store, see Chapter 13, *High Availability*.

10.5.1. Oracle BDB JE download

For details, see Section 10.4.1, “Oracle BDB JE download”.

10.5.2. Oracle BDB JE jar installation

For details, see Section 10.4.2, “Oracle BDB JE jar installation”.

10.5.3. Configuration

In order to use the BDBHAMessageStore, you must use a Virtual Host XML configuration file when defining a VirtualHost, configuring it for each VirtualHost desired by updating the store element to specify the associated store class, provide a directory location for the data to be written, and configure the replication group and policies used by BDB JA HA.

A general configuration example is shown here, however it is strongly recommended you examine the wider context of Chapter 13, *High Availability* for a fuller discussion of the various configuration options and how to use them.

Chapter 11. Security

11.1. Authentication Providers

In order to successfully establish a connection to the Java Broker, the connection must be authenticated. The Java Broker supports a number of different authentication schemes, each with its own "authentication provider". Any number of Authentication Providers can be configured on the Broker at the same time.

The Authentication Providers can be configured using REST Management interfaces and Web Management Console.

The following Authentication Provider managing operations are available from Web Management Console:

- A new Authentication Provider can be added by clicking onto "Add Provider" on the Broker tab.
- An Authentication Provider details can be viewed on the Authentication Provider tab. The tab is displayed after clicking onto Authentication Provider name in the Broker object tree or after clicking onto Authentication Provider row in Authentication Providers grid on the Broker tab.
- Editing of Authentication Provider can be performed by clicking on "Edit" button on Authentication Provider tab.
- An existing Authentication Provider can be deleted by clicking on "Delete Provider" button on Broker tab or "Delete" button on the Authentication Provider tab.

The Authentication Provider type and name cannot be changed for existing providers as editing of name and type is unsupported at the moment. Only provider specific attributes can be modified in the editing dialog and stored in the broker configuration store.

Important

Only unused Authentication Provider can be deleted. For delete requests attempting to delete Authentication Provider associated with the Ports, the errors will be returned and delete operations will be aborted. It is possible to change the Authentication Provider on Port at runtime. However, the Broker restart is required for changes on Port to take effect.

11.1.1. Simple LDAP Authentication Provider

SimpleLDAPAuthenticationProvider authenticates connections against a Directory (LDAP).

To create a SimpleLDAPAuthenticationProvider the following mandatory fields are required:

- *LDAP server URL* is the URL of the server, for example, `ldaps://example.com:636`
- *Search context* is the distinguished name of the search base object. It defines the location from which the search for users begins, for example, `dc=users,dc=example,dc=com`
- *Search filter* is a DN template to find an LDAP user entry by provided user name, for example, `(uid={0})`

Additionally, the following optional fields can be specified:

- *LDAP context factory* is a fully qualified class name for the JNDI LDAP context factory. This class must implement the InitialContextFactory [http://docs.oracle.com/javase/6/docs/api/javax/naming/spi/InitialContextFactory.html] interface and produce instances of DirContext [http://docs.oracle.com/

javase/6/docs/api/javax/naming/directory/DirContext.html]. If not specified a default value of `com.sun.jndi.ldap.LdapCtxFactory` is used.

- *LDAP authentication URL* is the URL of LDAP server for performing "ldap bind". If not specified, the *LDAP server URL* will be used for both searches and authentications.
- *Truststore name* is a name of configured truststore. Use this if connecting to a Directory over SSL (i.e. ldaps://) which is protected by a certificate signed by a private CA (or utilising a self-signed certificate).

Important

In order to protect the security of the user's password, when using LDAP authentication, you must:

- Use SSL on the broker's AMQP, JMX, and HTTP ports to protect the password during transmission to the Broker.
- Authenticate to the Directory using SSL (i.e. ldaps://) to protect the password during transmission from the Broker to the Directory.

The LDAP Authentication Provider works in the following manner. It first connects to the Directory anonymously and searches for the ldap entity which is identified by the username. The search begins at the distinguished name identified by `Search Context` and uses the username as a filter. The search scope is sub-tree meaning the search will include the base object and the subtree extending beneath it.

If the search returns a match, the Authentication Provider then attempts to bind to the LDAP server with the given name and the password. Note that simple security authentication [http://docs.oracle.com/javase/6/docs/api/javax/naming/Context.html#SECURITY_AUTHENTICATION] is used so the Directory receives the password in the clear.

11.1.2. Kerberos

Kereberos Authentication Provider uses java GSS-API SASL mechanism to authenticate the connections.

Configuration of kerberos is done through system properties (there doesn't seem to be a way around this unfortunately).

```
    export JAVA_OPTS=-Djavax.security.auth.useSubjectCredsOnly=false -Djava.securi
    ${QPID_HOME}/bin/qpid-server
```

Where qpid.conf would look something like this:

```
com.sun.security.jgss.accept {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    storeKey=true
    doNotPrompt=true
    realm="EXAMPLE.COM"
    useSubjectCredsOnly=false
    kdc="kerberos.example.com"
    keyTab="/path/to/keytab-file"
    principal="<name>/<host>";
};
```

Where realm, kdc, keyTab and principal should obviously be set correctly for the environment where you are running (see the existing documentation for the C++ broker about creating a keytab file).

Note: You may need to install the "Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files" appropriate for your JDK in order to get Kerberos support working.

Since Kerberos support only works where SASL authentication is available (e.g. not for JMX authentication) you may wish to also include an alternative Authentication Provider configuration, and use this for JMX and HTTP ports.

11.1.3. External (SSL Client Certificates)

When requiring SSL Client Certificates be presented the External Authentication Provider can be used, such that the user is authenticated based on trust of their certificate alone, and the X500Principal from the SSL session is then used as the username for the connection, instead of also requiring the user to present a valid username and password.

Note: The External Authentication Provider should typically only be used on the AMQP ports, in conjunction with SSL client certificate authentication. It is not intended for other uses such as the JMX management port and will treat any non-sasl authentication processes on these ports as successful with the given username. As such you should configure another Authentication Provider for use on non-AMQP ports. Perhaps the only exception to this would be where the broker is embedded in a container that is itself externally protecting the HTTP interface and then providing the remote users name.

On creation of External Provider the use of full DN or username CN as a principal name can be configured. If field "Use the full DN as the Username" is set to "true" the full DN is used as an authenticated principal name. If field "Use the full DN as the Username" is set to "false" the user name CN part is used as the authenticated principal name. Setting the field to "false" is particular useful when ACL is required, as at the moment, ACL does not support commas in the user name.

11.1.4. Anonymous

The Anonymous Authentication Provider will allow users to connect with or without credentials and result in their identification on the broker as the user ANONYMOUS. This Provider does not require specification of any additional fields on creation.

11.1.5. Plain Password File

The PlainPasswordFile Provider uses local file to store and manage user credentials. When creating an authentication provider the path to the file needs to be specified. If specified file does not exist an empty file is created automatically on Authentication Provider creation. On Provider deletion the password file is deleted as well. For this Provider user credentials can be added, removed or changed using REST management interfaces and web management console.

On navigating to the Plain Password File Provider tab (by clicking onto provider name from Broker tree or provider row in providers grid on Broker tab) the list of existing credentials is displayed on the tab with the buttons "Add User" and "Delete Users" to add new user credentials and delete the existing user credentials respectively. On clicking into user name on Users grid the pop-up dialog to change the password is displayed.

11.1.5.1. Plain Password File Format

The user credentials are stored on the single file line as user name and user password pairs separated by colon character.

```
# password file format
# <user name>: <user password>
```

```
guest:guest
```

11.1.6. Base64MD5 Password File

Base64MD5PasswordFile Provider uses local file to store and manage user credentials similar to Similar to PlainPasswordFile but instead of storing a password the MD5 password digest encoded with Base64 encoding is stored in the file. When creating an authentication provider the path to the file needs to be specified. If specified file does not exist an empty file is created automatically on Authentication Provider creation. On Base64MD5PasswordFile Provider deletion the password file is deleted as well. For this Provider user credentials can be added, removed or changed using REST management interfaces and web management console.

On navigating to the Base64MD5PasswordFile Provider tab (by clicking onto provider name from Broker tree or provider row in providers grid on Broker tab) the list of existing credentials is displayed on the tab with the buttons "Add User" and "Delete Users" to add new user credentials and delete the existing user credentials respectively. On clicking into user name on Users grid the pop-up dialog to change the password is displayed.

11.2. Group Providers

The Java broker utilises GroupProviders to allow assigning users to groups for use in ACLs. Following authentication by a given Authentication Provider, the configured Group Providers are consulted allowing the assignment of GroupPrincipals for a given authenticated user. Any number of Group Providers can be added into the Broker. All of them will be checked for the presence of the groups for a given authenticated user.

The *Group Provider* can be configured using REST Management interfaces and Web Management Console.

The following *Group Provider* managing operations are available from Web Management Console:

- A new Group Provider can be added by clicking onto "Add Group Provider" button on a Broker tab.
- An existing providers can be removed by pressing "Delete Group Provider" button on Broker tab or Group Provider tab.
- On clicking onto provider name in the Group Providers grid or Broker object tree, the tab for the Group Provider is displayed.
- A new group can be added into the Group Provider by clicking onto "Add Group" button on provider tab.
- An existing group can be deleted from the Group Provider by clicking onto "Delete Group" button on provider tab.
- On clicking onto group name in the groups grid, the tab with the list of existing group members is displayed for the Group.
- From the Group tab a new member can be added into a group or existing members can be deleted from a group by clicking on "Add Group Member" or "Remove Group Members" accordingly.

11.2.1. GroupFile Provider

The *GroupFile* Provider allows specifying group membership in a flat file on disk. On adding a new GroupFile Provider the path to the groups file is required to be specified. If file does not exist an empty file is created automatically. On deletion of GroupFile Provider the groups file is deleted as well. Only one instance of "GroupFile" Provider per groups file location can be created. On attempt to create another GroupFile Provider pointing to the same location the error will be displayed and the creation will be aborted.

11.2.1.1. File Format

The groups file has the following format:

```

    # <GroupName>.users = <comma deliminated user list>
    # For example:

    administrators.users = admin,manager
```

Only users can be added to a group currently, not other groups. Usernames can't contain commas.

Lines starting with a '#' are treated as comments when opening the file, but these are not preserved when the broker updates the file due to changes made through the management interface.

11.3. Access Control Lists

In Qpid, Access Control Lists (ACLs) specify which actions can be performed by each authenticated user. To enable, an *Access Control Provider* needs to be configured on the *Broker* level or/and ACL configuration should be provided on a *Virtual Host* level. The first imposes the ACL broker wide, and the second is applied to individual virtual hosts. The *Access Control Provider* of type "AclFile" uses local file to specify the ACL rules. By convention, this file should have a .acl extension.

A Group Provider can be configured with ACL to define the user groups which can be used in ACL to determine the ACL rules applicable to the entire group. The configuration details for the Group Providers are described in Section 11.2, “Group Providers”. On creation of ACL Provider with group rules, the Group Provider should be added first. Otherwise, if the individual ACL rules are not defined for the logged principal the following invocation of management operations could be denied due to absence of the required groups.

Only one *Access Control Provider* can be used by the Broker. If several *Access Control Providers* are configured on Broker level only one of them will be used (the latest one). Section 14.2, “Configuring ACL” shows how to configure ACL on *Virtual Host* using virtual host configuration xml. If both Broker *Access Control Provider* and *Virtual Host* ACL are configured, the *Virtual Host* ACL is used for authorization of operations on *Virtual Host* and Virtual Host objects and Broker level ACL is used to authorization of operations on Broker and Broker children (excluding Virtual Hosts having ACL configured).

The ACL Providers can be configured using REST Management interfaces and Web Management Console.

The following ACL Provider managing operations are available from Web Management Console:

- A new ACL Provider can be added by clicking onto "Add Access Control Provider" on the Broker tab.
- An ACL Provider details can be viewed on the Access Control Provider tab. The tab is shown after clicking onto ACL Provider name in the Broker object tree or after clicking onto ACL Provider row in ACL Providers grid on the Broker tab.
- An existing ACL Provider can be deleted by clicking onto buttons "Delete Access Control Provider" on the Broker tab or Access Control Provider tab.

11.3.1. Writing .acl files

The ACL file consists of a series of rules associating behaviour for a user or group. Use of groups can serve to make the ACL file more concise. See Configuring Group Providers for more information on defining groups.

Each ACL rule grants or denies a particular action on an object to a user/group. The rule may be augmented with one or more properties, restricting the rule's applicability.

```
      ACL ALLOW alice CREATE QUEUE              # Grants alice permission to creat
      ACL DENY bob CREATE QUEUE name="myqueue"  # Denies bob permission to create
```

The ACL is considered in strict line order with the first matching rule taking precedence over all those that follow. In the following example, if the user bob tries to create an exchange "myexch", the operation will be allowed by the first rule. The second rule will never be considered.

```
      ACL ALLOW bob ALL EXCHANGE
      ACL DENY bob CREATE EXCHANGE name="myexch"  # Dead rule
```

If the desire is to allow bob to create all exchanges except "myexch", order of the rules must be reversed:

```
      ACL DENY bob CREATE EXCHANGE name="myexch"
      ACL ALLOW bob ALL EXCHANGE
```

All ACL files end with an implict rule denying all operations to all users. It is as if each file ends with

```
ACL DENY ALL ALL
```

If instead you wish to *allow* all operations other than those controlled by earlier rules, add

```
ACL ALLOW ALL ALL
```

to the bottom of the ACL file.

When writing a new ACL, a good approach is to begin with an .acl file containing only

```
ACL DENY-LOG ALL ALL
```

which will cause the Broker to deny all operations with details of the denial logged to the Qpid log file. Build up the ACL rule by rule, gradually working through the use-cases of your system. Once the ACL is complete, consider switching the DENY-LOG actions to DENY to improve performamce and reduce log noise.

ACL rules are very powerful: it is possible to write very granular rules specifying many broker objects and their properties. Most projects probably won't need this degree of flexibility. A reasonable approach is to choose to apply permissions at a certain level of abstraction (e.g. QUEUE) and apply them consistently across the whole system.

11.3.2. Syntax

ACL rules follow this syntax:

```
ACL {permission} {<group-name>|<user-name>>|ALL} {action|ALL} [object|ALL] [p
```

Comments may be introduced with the hash (#) character and are ignored. Long lines can be broken with the slash (\) character.

```
# A comment
ACL ALLOW admin CREATE ALL # Also a comment
ACL DENY guest \
ALL ALL   # A broken line
```

Table 11.1. List of ACL permission

ALLOW	Allow the action
ALLOW-LOG	Allow the action and log the action in the log
DENY	Deny the action
DENY-LOG	Deny the action and log the action in the log

Table 11.2. List of ACL actions

CONSUME	Applied when subscriptions are created
PUBLISH	Applied on a per message basis on publish message transfers
CREATE	Applied when an object is created, such as bindings, queues, exchanges
ACCESS	Applied when an object is read or accessed
BIND	Applied when queues are bound to exchanges
UNBIND	Applied when queues are unbound from exchanges
DELETE	Applied when objects are deleted
PURGE	Applied when purge the contents of a queue
UPDATE	Applied when an object is updated
CONFIGURE	Applied when an object is configured via REST management interfaces(Java Broker only).

Table 11.3. List of ACL objects

VIRTUALHOST	A virtualhost (Java Broker only)
MANAGEMENT	Management - for web and JMX (Java Broker only)
QUEUE	A queue
EXCHANGE	An exchange
USER	A user (Java Broker only)
GROUP	A group (Java Broker only)
METHOD	Management or agent or broker method (Java Broker only)

LINK	A federation or inter-broker link (not currently used in Java Broker)
BROKER	The broker

Table 11.4. List of ACL properties

name	String. Object name, such as a queue name, exchange name or JMX method name.
durable	Boolean. Indicates the object is durable
routingkey	String. Specifies routing key
passive	Boolean. Indicates the presence of a *`passive`* flag
autodelete	Boolean. Indicates whether or not the object gets deleted when the connection is closed
exclusive	Boolean. Indicates the presence of an *`exclusive`* flag
temporary	Boolean. Indicates the presence of an *`temporary`* 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`*
component	String. JMX component name (Java Broker only)
schemapackage	String. QMF schema package name (Not used in Java Broker)
schemaclass	String. QMF schema class name (Not used in Java Broker)
from_network	Comma-separated strings representing IPv4 address ranges. Intended for use in ACCESS VIRTUALHOST rules to apply firewall-like restrictions. The rule matches if any of the address ranges match the IPv4 address of the messaging client. The address ranges are specified using either Classless Inter-Domain Routing notation (e.g. 192.168.1.0/24; see RFC 4632 [http://tools.ietf.org/html/rfc4632]) or wildcards (e.g. 192.169.1.*). Java Broker only.
from_hostname	Comma-separated strings representing hostnames, specified using Perl-style regular expressions, e.g. .*\.example\.company\.com Intended for use in ACCESS VIRTUALHOST rules to apply firewall-like restrictions. The rule matches if any of the patterns match the hostname of the messaging client.

	To look up the client's hostname, Qpid uses Java's DNS support, which internally caches its results. You can modify the time-to-live of cached results using the *.ttl properties described on the Java Networking Properties [http://docs.oracle.com/javase/6/docs/technotes/guides/net/properties.html] page. For example, you can either set system property sun.net.inetaddr.ttl from the command line (e.g. export QPID_OPTS="-Dsun.net.inetaddr.ttl=0") or networkaddress.cache.ttl in $JAVA_HOME/lib/security/java.security. The latter is preferred because it is JVM vendor-independent. Java Broker only.

Table 11.5. List of ACL rules

UserManagement	User maintainance; create/delete/view users, change passwords etc	permissionable at broker level only
ConfigurationManagement	Dynammically reload configuration from disk.	permissionable at broker level only
LoggingManagement	Dynammically control Qpid logging level	permissionable at broker level only
ServerInformation	Read-only information regarding the Qpid: version number etc	permissionable at broker level only
VirtualHost.Queue	Queue maintainance; copy/move/purge/view etc	
VirtualHost.Exchange	Exchange maintenance; bind/unbind queues to exchanges	
VirtualHost.VirtualHost	Virtual host maintainace; create/delete exchanges, queues etc	

11.3.3. Worked Examples

Here are some example ACLs illustrating common use cases. In addition, note that the Java broker provides a complete example ACL file, located at etc/broker_example.acl.

11.3.3.1. Worked example 1 - Management rights

Suppose you wish to permission two users: a user 'operator' must be able to perform all Management operations, and a user 'readonly' must be enable to perform only read-only functions. Neither 'operator' nor 'readonly' should be allowed to connect clients for messaging.

```
# Deny (loggged) operator/readonly permission to connect messaging clients.
ACL DENY-LOG operator ACCESS VIRTUALHOST
ACL DENY-LOG readonly ACCESS VIRTUALHOST
# Give operator permission to perfom all other actions
```

```
ACL ALLOW operator ALL ALL
# Give readonly permission to execute only read-only actions
ACL ALLOW readonly ACCESS ALL
...
... rules for other users
...
# Explicitly deny all (log) to eveyone
ACL DENY-LOG ALL ALL
```

11.3.3.2. Worked example 2 - User maintainer group

Suppose you wish to restrict User Management operations to users belonging to a group 'usermaint'. No other user is allowed to perform user maintainence This example illustrates the permissioning of an individual component.

```
# Give usermaint access to management and permission to execute all JMX Methods on
# UserManagement MBean and perform all actions for USER objects
ACL ALLOW usermaint ACCESS MANAGEMENT
ACL ALLOW usermaint ALL METHOD component="UserManagement"
ACL ALLOW usermaint ALL USER
ACL DENY ALL ALL METHOD component="UserManagement"
ACL DENY ALL ALL USER
...
... rules for other users
...
ACL DENY-LOG ALL ALL
```

11.3.3.3. Worked example 3 - Request/Response messaging

Suppose you wish to permission a system using a request/response paradigm. Two users: 'client' publishes requests; 'server' consumes the requests and generates a response. This example illustrates the permissioning of AMQP exchanges and queues.

```
# Allow client and server to connect to the virtual host.
ACL ALLOW client ACCESS VIRTUALHOST
ACL ALLOW server ACCESS VIRTUALHOST

# Client side
# Allow the 'client' user to publish requests to the request queue. As is the norm
# is required to create a temporary queue on which the server will respond.  Conse
# of the temporary queues and consumption of messages from it.
ACL ALLOW client CREATE QUEUE temporary="true"
ACL ALLOW client CONSUME QUEUE temporary="true"
ACL ALLOW client DELETE QUEUE temporary="true"
ACL ALLOW client BIND EXCHANGE name="amq.direct" temporary="true"
ACL ALLOW client UNBIND EXCHANGE name="amq.direct" temporary="true"
ACL ALLOW client PUBLISH EXCHANGE name="amq.direct" routingKey="example.RequestQue

# Server side
# Allow the 'server' user to consume from the request queue and publish a response
# client.  We also allow the server to create the request queue.
ACL ALLOW server CREATE QUEUE name="example.RequestQueue"
ACL ALLOW server CONSUME QUEUE name="example.RequestQueue"
ACL ALLOW server BIND EXCHANGE
ACL ALLOW server PUBLISH EXCHANGE name="amq.direct" routingKey="TempQueue*"

ACL DENY-LOG all all
```

11.3.3.4. Worked example 4 - firewall-like access control

This example illustrates how to set up an ACL that restricts the IP addresses and hostnames of messaging clients that can access a virtual host.

```
################
# Hostname rules
################

# Allow messaging clients from company1.com and company1.co.uk to connect
ACL ALLOW all ACCESS VIRTUALHOST from_hostname=".*\.company1\.com,.*\.company1\.co

# Deny messaging clients from hosts within the dev subdomain
ACL DENY-LOG all ACCESS VIRTUALHOST from_hostname=".*\.dev\.company1\.com"

##################
# IP address rules
##################

# Deny access to all users in the IP ranges 192.168.1.0-192.168.1.255 and 192.168.
# using the notation specified in RFC 4632, "Classless Inter-domain Routing (CIDR)
ACL DENY-LOG messaging-users ACCESS VIRTUALHOST \
  from_network="192.168.1.0/24,192.168.2.0/24"

# Deny access to all users in the IP ranges 192.169.1.0-192.169.1.255 and 192.169.
# using wildcard notation.
ACL DENY-LOG messaging-users ACCESS VIRTUALHOST \
  from_network="192.169.1.*,192.169.2.*"

ACL DENY-LOG all all
```

11.3.3.5. Worked example 5 - REST management ACL example

This example illustrates how to set up an ACL that restricts usage of REST management interfaces.

```
# allow to the users from webadmins group to change broker model
# this rule allows adding/removing/editing of Broker level objects:
# Broker, Virtual Host, Group Provider, Authentication Provider, Port, Access Cont
ACL ALLOW-LOG webadmins CONFIGURE BROKER

# allow to the users from webadmins group to perform
# create/update/delete on Virtual Host children
```

```
ACL ALLOW-LOG webadmins CREATE QUEUE
ACL ALLOW-LOG webadmins UPDATE QUEUE
ACL ALLOW-LOG webadmins DELETE QUEUE
ACL ALLOW-LOG webadmins PURGE  QUEUE
ACL ALLOW-LOG webadmins CREATE EXCHANGE
ACL ALLOW-LOG webadmins DELETE EXCHANGE
ACL ALLOW-LOG webadmins BIND   EXCHANGE
ACL ALLOW-LOG webadmins UNBIND EXCHANGE

# allow to the users from webadmins group to create/update/delete groups on Group
ACL ALLOW-LOG webadmins CREATE GROUP
ACL ALLOW-LOG webadmins DELETE GROUP
ACL ALLOW-LOG webadmins UPDATE GROUP

# allow to the users from webadmins group to create/update/delete users for Authen
ACL ALLOW-LOG webadmins CREATE USER
ACL ALLOW-LOG webadmins DELETE USER
ACL ALLOW-LOG webadmins UPDATE USER

# allow to the users from webadmins group to move, copy and delete messagaes
# using REST management interfaces
ACL ALLOW-LOG webadmins UPDATE METHOD

# at the moment only the following UPDATE METHOD rules are supported by web manage
#ACL ALLOW-LOG webadmins UPDATE METHOD component="VirtualHost.Queue" name="moveMes
#ACL ALLOW-LOG webadmins UPDATE METHOD component="VirtualHost.Queue" name="copyMes
#ACL ALLOW-LOG webadmins UPDATE METHOD component="VirtualHost.Queue" name="deleteM

ACL DENY-LOG all all
```

11.4. SSL

This section guides through the details of configuration of Keystores and Trsustores required for enabling of SSL transport and Client Certificate Authentication on Broker ports. The details how to configure SSL on Broker ports are provided in Chapter 6, *Broker Ports*.

11.4.1. Keystore Configuration

A Keystore can be added/deleted/edited using REST Management interfaces and Web Management Console. Any number of Keystores can be configured on the Broker. SSL ports can be configured with different Keystores.

The following Keystore managing operations are available from Web Management Console:

- A new Keystore can be added by clicking on "Add Key Store" button on the Broker tab.
- Keystore details can be viewed on the Keystore tab which is displayed after clicking on Keystore name in the Broker object tree or after clicking on Keystore row in Keystores grid on the Broker tab.
- Editing of Keystore can be performed by clicking on "Edit" button on the Keystore tab. Changing of Keystore name is unsupported at the moment. If changed Keystore is used by the Port the changes on Port object will take effect after Broker restart.

- An existing Keystore can be deleted by clicking on "Delete Key Store" button on Broker tab or hitting "Delete" button on the Keystore tab. Only unused Keystores can be deleted. The deletion of the Keystore configured on any Broker Port is not allowed.

The "Keystore certificate alias" field is an optional way of specifying which certificate the broker should use if the keystore contains multiple entries. Optionally "Key manager factory algorithm" and "Key store type" can be specified on Keystore creation.

Important

The password of the certificate used by the Broker **must** match the password of the keystore itself. This is a restriction of the Qpid Broker implementation. If using the keytool [http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html] utility, note that this means the argument to the `-keypass` option must match the `-storepass` option.

11.4.2. Truststore / Client Certificate Authentication

The SSL trustore and related Client Certificate Authentication behaviour can be configured by adding a Trustore configured object and associating it with the SSL port. A Truststore can be added/deleted/edited using REST Management interfaces and Web Management Console. Any number of Trustores can be configured on the Broker. Multiple Trustores can be configured on Broker SSL Ports.

The following Truststore managing operations are available from Web Management Console:

- A new Truststore can be added by clicking on "Add Trust Store" button on the Broker tab.
- Truststore details can be viewed on the Truststore tab which is displayed after clicking onto Truststore name in the Broker object tree or after clicking onto Truststore row in Truststores grid on the Broker tab.
- Trustore can be edited by clicking onto "Edit" button on the Trustore tab. Changing of Trustore name is unsupported at the moment.
- An existing Trustore can be deleted by clicking onto "Delete Trust Store" button on Broker tab or "Delete" button on the Truststore tab. Only unused Truststores can be deleted. The deletion of the Truststore configured on any Broker Port is not allowed.

When "Peers Only" option is selected for the Truststore it will allow logging in for the clients with the certificate exactly matching the certificate loaded in the Truststore database, thus, authenticating the connections with self signed certificates not nessesary signed by CA.

"Trust manager factory algorithm" and "Trust store type" can be optionally specified for the Trustore.

Chapter 12. Runtime

12.1. Log Files

12.2. Alerts

12.3. Disk Space Management

12.3.1. Producer Flow Control

12.3.1.1. General Information

The Qpid 0.6 release introduced a simplistic producer-side flow control mechanism into the Java Messaging Broker, causing producers to be flow-controlled when they attempt to send messages to an overfull queue. Qpid 0.18 introduced a similar mechanism triggered by an overfull persistent message store on a virtual host.

12.3.1.2. Server Configuration

12.3.1.2.1. Configuring a Queue to use flow control

Flow control is enabled on a producer when it sends a message to a Queue which is "overfull". The producer flow control will be rescinded when all Queues on which a producer is blocking become "underfull". A Queue is defined as overfull when the size (in bytes) of the messages on the queue exceeds the "capacity" of the Queue. A Queue becomes "underfull" when its size becomes less than the "flowResumeCapacity".

Examples how to configure flow control in virtual host configuration are provided in Section 14.11, “Configuring of Producer Flow Control”.

Where no flowResumeCapacity is set, the flowResumeCapacity is set to be equal to the capacity. Where no capacity is set, capacity is defaulted to 0 meaning there is no capacity limit.

Important

Flow control can be configured globally for all virtual hosts by specifying threshold values for Broker flow control attributes.

12.3.1.2.1.1. Broker Log Messages

There are four new Broker log messages that may occur if flow control through queue capacity limits is enabled. Firstly, when a capacity limited queue becomes overfull, a log message similar to the following is produced

```
MESSAGE [vh(/test)/qu(MyQueue)] [vh(/test)/qu(MyQueue)] QUE-1003 : Overfull : Size
```

Then for each channel which becomes blocked upon the overful queue a log message similar to the following is produced:

```
MESSAGE [con:2(guest@anonymous(713889609)/test)/ch:1] [con:2(guest@anonymous(71388
```

When enough messages have been consumed from the queue that it becomes underfull, then the following log is generated:

```
MESSAGE [vh(/test)/qu(MyQueue)] [vh(/test)/qu(MyQueue)] QUE-1004 : Underfull : Siz
```

And for every channel which becomes unblocked you will see a message similar to:

```
MESSAGE [con:2(guest@anonymous(713889609)/test)/ch:1] [con:2(guest@anonymous(71388
```

Obviously the details of connection, virtual host, queue, size, capacity, etc would depend on the configuration in use.

12.3.1.2.2. Disk quota-based flow control

Since version 0.18 of Qpid Broker, flow control can be triggered when a configured disk quota is exceeded. This is supported by the BDB and Derby message stores.

This functionality blocks all producers on reaching the disk overflow limit. When consumers consume the messages, causing disk space usage to falls below the underflow limit, the producers are unblocked and continue working as normal.

Two limits can be configured:

overfull limit - the maximum space on disk (in bytes) which can be used by store.

underfull limit - when the space on disk drops below this limit, producers are allowed to resume publishing.

An example how to configure disk quota-based flow control in virtual host configuration is provided in Section 14.12, “Configuring of Disk Quota-based Flow Control”.

The disk quota functionality is based on "best effort" principle. This means the broker cannot guarantee that the disk space limit will not be exceeded. If several concurrent transactions are started before the limit is reached, which collectively cause the limit to be exceeded, the broker may allow all of them to be committed.

12.3.1.2.2.1. Broker Log Messages for quota flow control

There are 2 new broker log messages that may occur if flow control through disk quota limits is enabled. When the virtual host is blocked due to exceeding of the disk quota limit the following message appears in the broker log

```
[vh(/test)/ms(BDBMessageStore)] MST-1008 : Store overfull, flow control will be en
```

When virtual host is unblocked after cleaning the disk space the following message appears in the broker log

```
[vh(/test)/ms(BDBMessageStore)] MST-1009 : Store overfull condition cleared
```

12.3.1.3. Client impact and configuration

If a producer sends to a queue which is overfull, the broker will respond by instructing the client not to send any more messages. The impact of this is that any future attempts to send will block until the broker rescinds the flow control order.

While blocking the client will periodically log the fact that it is blocked waiting on flow control.

```
WARN   Message send delayed by 5s due to broker enforced flow control
WARN   Message send delayed by 10s due to broker enforced flow control
```

After a set period the send will timeout and throw a JMSException to the calling code.

If such a JMSException is thrown, the message will not be sent to the broker, however the underlying Session may still be active - in particular if the Session is transactional then the current transaction will not be automatically rolled back. Users may choose to either attempt to resend the message, or to roll back any transactional work and close the Session.

Both the timeout delay and the periodicity of the warning messages can be set using Java system properties.

The amount of time (in milliseconds) to wait before timing out is controlled by the property qpid.flow_control_wait_failure.

The frequency at which the log message informing that the producer is flow controlled is sent is controlled by the system property qpid.flow_control_wait_notify_period.

Adding the following to the command line to start the client would result in a timeout of one minute, with warning messages every ten seconds:

```
-Dqpid.flow_control_wait_failure=60000
-Dqpid.flow_control_wait_notify_period=10000
```

12.3.1.3.1. Older Clients

The flow control feature was first added to the Java broker/client in the 0.6 release. If an older client connects to the broker then the flow control commands will be ignored by it and it will not be blocked. So to fully benefit from this feature both Client and Broker need to be at least version 0.6.

12.4. Producer Transaction Timeout

12.4.1. General Information

The transaction timeout mechanism is used to control broker resources when clients producing messages using transactional sessions hang or otherwise become unresponsive, or simply begin a transaction and keep using it without ever calling Session#commit() [http://docs.oracle.com/javaee/6/api/javax/jms/Session.html#commit].

Users can choose to configure an idleWarn or openWarn threshold, after which the identified transaction should be logged as a WARN level alert as well as (more importantly) an idleClose or openClose threshold after which the transaction and the connection it applies to will be closed.

This feature is particularly useful in environments where the owner of the broker does not have full control over the implementation of clients, such as in a shared services deployment.

The following section provide more details on this feature and its use.

12.4.2. Purpose

This feature has been introduced to address the scenario where an open transaction on the broker holds an open transaction on the persistent store. This can have undesirable consequences if the store does not time out or close long-running transactions, such as with BDB. This can can result in a rapid increase in disk usage size, bounded only by available space, due to growth of the transaction log.

12.4.3. Scope

Note that only MessageProducer [http://docs.oracle.com/javaee/6/api/javax/jms/MessageProducer.html] clients will be affected by a transaction timeout, since store transaction lifespan on a consumer only spans the execution of the call to Session#commit() and there is no scope for a long-lived transaction to arise.

It is also important to note that the transaction timeout mechanism is purely a JMS transaction timeout, and unrelated to any other timeouts in the Qpid client library and will have no impact on any RDBMS your application may utilise.

12.4.4. Effect

Full details of configuration options are provided in the sections that follow. This section gives a brief overview of what the Transaction Timeout feature can do.

12.4.4.1. Broker Logging and Connection Close

When the openWarn or idleWarn specified threshold is exceeded, the broker will log a WARN level alert with details of the connection and channel on which the threshold has been exceeded, along with the age of the transaction.

When the openClose or idleClose specified threshold value is exceeded, the broker will throw an exception back to the client connection via the ExceptionListener [http://docs.oracle.com/javaee/6/api/javax/jms/ExceptionListener.html], log the action and then close the connection.

The example broker log output shown below is where the idleWarn threshold specified is lower than the idleClose threshold and the broker therefore logs the idle transaction 3 times before the close threshold is triggered and the connection closed out.

```
CHN-1008 : Idle Transaction : 13,116 ms
CHN-1008 : Idle Transaction : 14,116 ms
CHN-1008 : Idle Transaction : 15,118 ms
CHN-1003 : Close
```

The second example broker log output shown below illustrates the same mechanism operating on an open transaction.

```
CHN-1007 : Open Transaction : 12,406 ms
CHN-1007 : Open Transaction : 13,406 ms
```

```
CHN-1007 : Open Transaction : 14,406 ms
CHN-1003 : Close
```

12.4.4.2. Client Side Effect

After a Close threshold has been exceeded, the trigger client will receive this exception on its exception listener [http://docs.oracle.com/javaee/6/api/javax/jms/ExceptionListener.html], prior to being disconnected:
`org.apache.qpid.AMQConnectionClosedException: Error: Idle transaction timed out [error code 506: resource error]`

Any later attempt to use the connection will result in this exception being thrown:

```
Producer: Caught an Exception: javax.jms.IllegalStateException: Object org.apache.
    javax.jms.IllegalStateException: Object org.apache.qpid.client.AMQSession_0_8@
    at org.apache.qpid.client.Closeable.checkNotClosed(Closeable.java:70)
    at org.apache.qpid.client.AMQSession.checkNotClosed(AMQSession.java:555)
    at org.apache.qpid.client.AMQSession.createBytesMessage(AMQSession.java:573)
```

Thus clients must be able to handle this case successfully, reconnecting where required and registering an exception listener on all connections. This is critical, and must be communicated to client applications by any broker owner switching on transaction timeouts.

12.4.5. Configuration

12.4.5.1. Configuration

Important

Transaction timeouts can be configured globally for all virtual hosts by setting corresponding Broker transaction timeout attributes.

Transaction timeouts can be configured separately on each defined virtual host, using the virtualhosts.xml file.

We would recommend that only warnings are configured at first, which should allow broker administrators to obtain an idea of the distribution of transaction lengths on their systems, and configure production settings appropriately for both warning and closure. Ideally establishing thresholds should be achieved in a representative UAT environment, with clients and broker running, prior to any production deployment.

It is impossible to give suggested values, due to the large variation in usage depending on the applications using a broker. However, clearly transactions should not span the expected lifetime of any client application as this would indicate a hung client.

When configuring warning and closure timeouts, it should be noted that these only apply to message producers that are connected to the broker, but that a timeout will cause the connection to be closed - this disconnecting all producers and consumers created on that connection.

This should not be an issue for environments using Mule or Spring, where connection factories can be configured appropriately to manage a single MessageProducer object per JMS Session and Connection. Clients that use the JMS API directly should be aware that sessions managing both consumers and producers, or multiple producers, will be affected by a single producer hanging or leaving a transaction idle or open, and closed, and must take appropriate action to handle that scenario.

12.4.5.2. Virtualhost configuration

The details how to configure Transaction Timeouts in Virtual Host configuration file are provided in Section 14.13, “Configuring Transaction Timeouts”

12.5. Handing Undeliverable Messages

12.5.1. Introduction

Messages that cannot be delivered successfully to a consumer (for instance, because the client is using a transacted session and rolls-back the transaction) can be made available on the queue again and then subsequently be redelivered, depending on the precise session acknowledgement mode and messaging model used by the application. This is normally desirable behaviour that contributes to the ability of a system to withstand unexpected errors. However, it leaves open the possibility for a message to be repeatedly redelivered (potentially indefinitely), consuming system resources and preventing the delivery of other messages. Such undeliverable messages are sometimes known as poison messages.

For an example, consider a stock ticker application that has been designed to consume prices contained within JMS TextMessages. What if inadvertently a BytesMessage is placed onto the queue? As the ticker application does not expect the BytesMessage, its processing might fail and cause it to roll-back the transaction, however the default behavior of the Broker would mean that the BytesMessage would be delivered over and over again, preventing the delivery of other legitimate messages, until an operator intervenes and removes the erroneous message from the queue.

Qpid has maximum delivery count and dead-letter queue (DLQ) features which can be used in concert to construct a system that automatically handles such a condition. These features are described in the following sections.

12.5.2. Maximum Delivery Count

Maximum delivery count is a property of a queue. If a consumer application is unable to process a message more than the specified number of times, then the broker will either route the message to a dead-letter queue (if one has been defined), or will discard the message.

In order for a maximum delivery count to be enforced, the consuming client *must* call Session#rollback() [http://docs.oracle.com/javaee/6/api/javax/jms/Session.html#rollback()] (or Session#recover() [http://docs.oracle.com/javaee/6/api/javax/jms/Session.html#recover()] if the session is not transacted). It is during the Broker's processing of Session#rollback() (or Session#recover()) that if a message has been seen at least the maximum number of times then it will move the message to the DLQ or discard the message.

If the consuming client fails in another manner, for instance, closes the connection, the message will not be re-routed and consumer application will see the same poison message again once it reconnects.

If the consuming application is using AMQP 0-9-1, 0-9, or 0-8 protocols, it is necessary to set the client system property `qpid.reject.behaviour` or connection or binding URL option `rejectbehaviour` to the value `system`.

It is possible to determine the number of times a message has been sent to a consumer via the Management interfaces, but is not possible to determine this information from a message client. Specifically, the optional JMS message header JMSXDeliveryCount is not supported.

Maximum Delivery Count can be enabled via management (see Chapter 5, *Configuring And Managing*) using the the queue declare property x-qpid-maximum-delivery-count or via configuration as illustrated below.

12.5.3. Dead Letter Queues (DLQ)

A Dead Letter Queue (DLQ) acts as an destination for messages that have somehow exceeded the normal bounds of processing and is utilised to prevent disruption to flow of other messages. When a DLQ is enabled for a given queue if a consuming client indicates it no longer wishes the receive the message (typically by exceeding a Maximum Delivery Count) then the message is moved onto the DLQ and removed from the original queue.

The DLQ feature causes generation of a Dead Letter Exchange and a Dead Letter Queue. These are named convention QueueName*_DLE* and QueueName*_DLQ*.

DLQs can be enabled via management (see Chapter 5, *Configuring And Managing*) using the queue declare property x-qpid-dlq-enabled or via configuration as illustrated below.

Avoid excessive queue depth

Applications making use of DLQs *should* make provision for the frequent examination of messages arriving on DLQs so that both corrective actions can be taken to resolve the underlying cause and organise for their timely removal from the DLQ. Messages on DLQs consume system resources in the same manner as messages on normal queues so excessive queue depths should not be permitted to develop.

12.5.4. Configuration

Important

DLQs/Maximum Delivery can be configured globally for all Virtual Hosts by specifying non-zero value for global Broker attribute "queue.maximumDeliveryAttempts" and setting of Broker attribute "queue.deadLetterQueueEnabled" to true.

An examples of configuring DLQs/Maximum Delivery Count using Virtual Hosts configuration file are described in Section 14.14, “Configuring DLQs/Maximum Delivery Count”.

12.6. Closing client connections on unroutable mandatory messages

12.6.1. Summary

Due to asynchronous nature of AMQP 0-8/0-9/0-9-1 protocols sending a message with a routing key for which no queue binding exist results in either message being bounced back (if it is mandatory or immediate) or discarded on broker side otherwise.

When a 'mandatory' message is returned back, the Qpid JMS client conveys this by delivering an *AMQNoRouteException* through the configured ExceptionListener on the Connection. This does not cause channel or connection closure, however it requires a special exception handling on client side in order to deal with *AMQNoRouteExceptions*. This could potentially be a problem when using various messaging frameworks (e.g Mule) as they usually close the connection on receiving any JMSException.

In order to simplify application handling of scenarios where 'mandatory' messages are being sent to queues which do not actually exist, the Java Broker can be configured such that it will respond to this situation by closing the connection rather than returning the unroutable message to the client as it normally should. From the application perspective, this will result in failure of synchronous operations in progress such as a session commit() call.

Note

This feature affects only transacted sessions.

Qpid JMS client sends 'mandatory' messages when using Queue destinations and 'non-mandatory' messages when using Topic destinations.

12.6.2. Configuring *closeWhenNoRoute*

The Broker attribute *closeWhenNoRoute* can be set to specify this feature on broker side. By default, it is turned on. Setting *closeWhenNoRoute* to *false* switches it off.

Setting the *closeWhenNoRoute* in the JMS client connection URL can override the broker configuration on a connection specific basis, for example :

Example 12.1. Disable feature to close connection on unroutable messages with client URL

```
amqp://guest:guest@clientid/?brokerlist='tcp://localhost:5672'&closeWhenNoRoute='f
```

If no value is specified on the client the broker setting will be used. If client setting is specified then it will take precedence over the broker-wide configuration. If the client specifies and broker does not support this feature the warning will be logged.

Chapter 13. High Availability

13.1. General Introduction

The term High Availability (HA) usually refers to having a number of instances of a service such as a Message Broker available so that should a service unexpectedly fail, or requires to be shutdown for maintenance, users may quickly connect to another instance and continue their work with minimal interuption. HA is one way to make a overall system more resilient by eliminating a single point of failure from a system.

HA offerings are usually categorised as **Active/Active** or **Active/Passive**. An Active/Active system is one where all nodes within the cluster are usuaully available for use by clients all of the time. In an Active/Passive system, one only node within the cluster is available for use by clients at any one time, whilst the others are in some kind of standby state, awaiting to quickly step-in in the event the active node becomes unavailable.

13.2. HA offerings of the Java Broker

The Java Broker's HA offering became available at release **0.18**. HA is provided by way of the HA features built into the Java Edition of the Berkley Database (BDB JE) [http://www.oracle.com/technetwork/products/berkeleydb/overview/index-093405.html] and as such is currently only available to Java Broker users who use the optional BDB JE based persistence store. This **optional** store requires the use of BDB JE which is licensed under the Sleepycat Licence, which is not compatible with the Apache Licence and thus BDB JE is not distributed with Qpid. Users who elect to use this optional store for the broker have to provide this dependency.

HA in the Java Broker provides an **Active/Passive** mode of operation with Virtual hosts being the unit of replication. The Active node (referred to as the **Master**) accepts all work from all the clients. The Passive nodes (referred to as **Replicas**) are unavailable for work: the only task they must perform is to remain in synch with the Master node by consuming a replication stream containing all data and state.

If the Master node fails, a Replica node is elected to become the new Master node. All clients automatically failover [1] to the new Master and continue their work.

The Java Broker HA solution is incompatible with the HA solution offered by the CPP Broker. It is not possible to co-locate Java and CPP Brokers within the same cluster.

HA is not currently available for those using the the **Derby Store** or **Memory Message Store**.

13.3. Two Node Cluster

13.3.1. Overview

In this HA solution, a cluster is formed with two nodes. one node serves as **master** and the other is a **replica**.

All data and state required for the operation of the virtual host is automatically sent from the master to the replica. This is called the replication stream. The master virtual host confirms each message is on the

[1]The automatic failover feature is available only for AMQP connections from the Java client. Management connections (JMX) do not current offer this feature.

replica before the client transaction completes. The exact way the client awaits for the master and replica is gorverned by the durability configuration, which is discussed later. In this way, the replica remains ready to take over the role of the master if the master becomes unavailable.

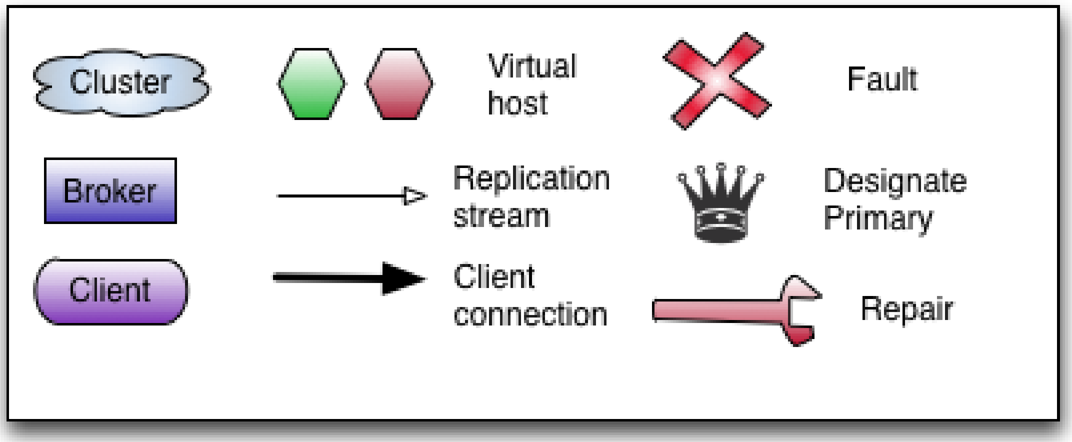
It is important to note that there is an inherent limitation of two node clusters is that the replica node cannot make itself master automatically in the event of master failure. This is because the replica has no way to distinguish between a network partition (with potentially the master still alive on the other side of the partition) and the case of genuine master failure. (If the replica were to elect itself as master, the cluster would run the risk of a split-brain [http://en.wikipedia.org/wiki/Split-brain_(computing)] scenario). In the event of a master failure, a third party must designate the replica as primary. This process is described in more detail later.

Clients connect to the cluster using a failover url. This allows the client to maintain a connection to the master in a way that is transparent to the client application.

13.3.2. Depictions of cluster operation

In this section, the operation of the cluster is depicted through a series of figures supported by explanatory text.

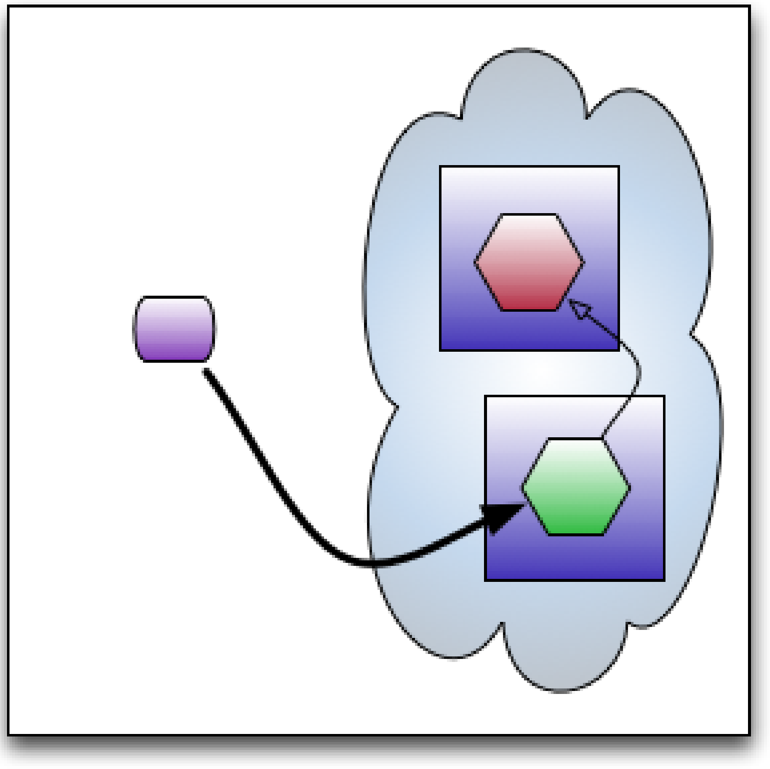
Figure 13.1. Key for figures



13.3.2.1. Normal Operation

The figure below illustrates normal operation. Clients connecting to the cluster by way of the failover URL achieve a connection to the master. As clients perform work (message production, consumption, queue creation etc), the master additionally sends this data to the replica over the network.

Figure 13.2. Normal operation of a two-node cluster



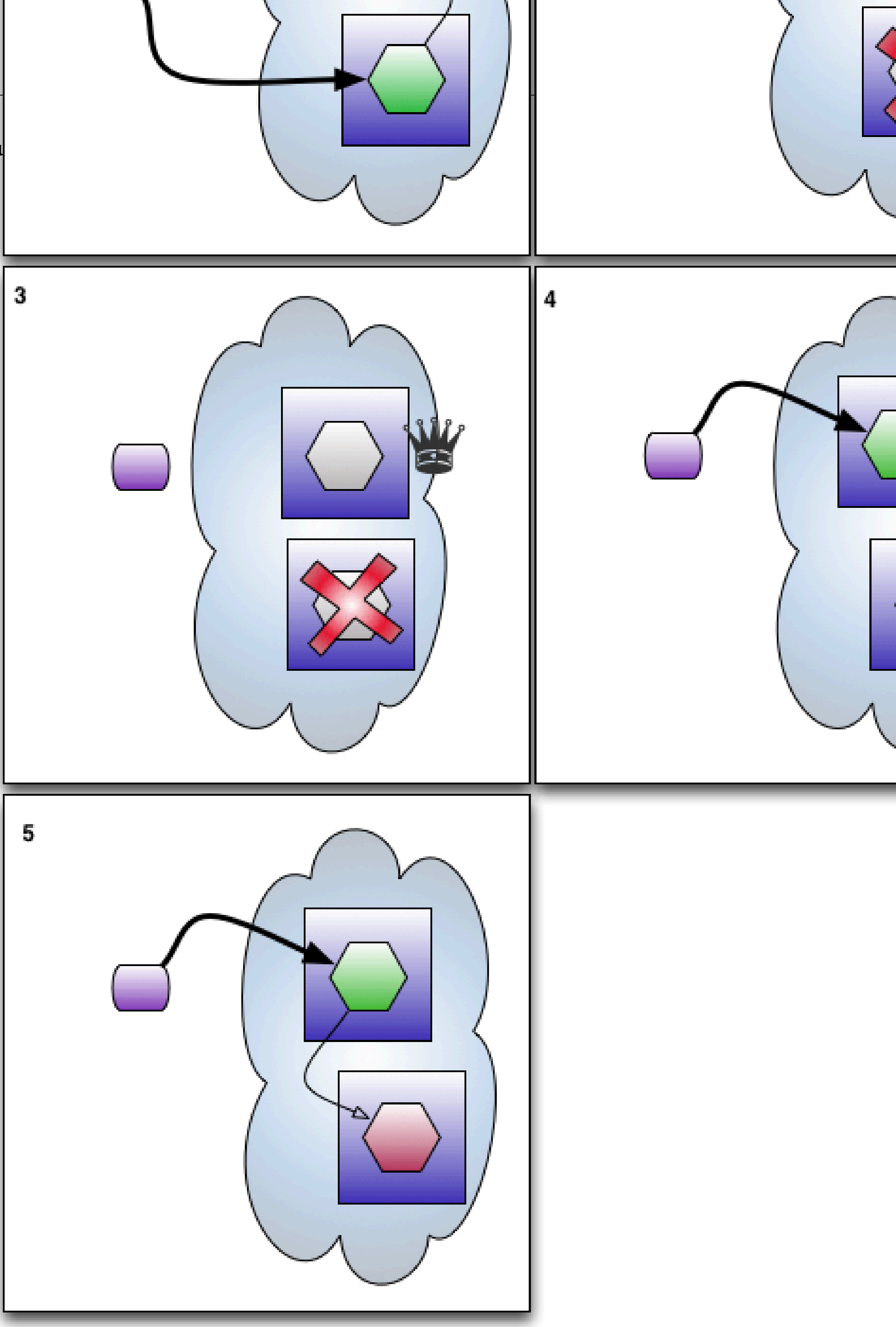
13.3.2.2. Master Failure and Recovery

The figure below illustrates a sequence of events whereby the master suffers a failure and the replica is made the master to allow the clients to continue to work. Later the old master is repaired and comes back on-line in replica role.

The item numbers in this list apply to the numbered boxes in the figure below.

1. System operating normally
2. Master suffers a failure and disconnects all clients. Replica realises that it is no longer in contact with master. Clients begin to try to reconnect to the cluster, although these connection attempts will fail at this point.
3. A third-party (an operator, a script or a combination of the two) verifies that the master has truely failed **and is no longer running**. If it has truely failed, the decision is made to designate the replica as primary, allowing it to assume the role of master despite the other node being down. This primary designation is performed using JMX.
4. Client connections to the new master succeed and the **service is restored** , albeit without a replica.
5. The old master is repaired and brought back on-line. It automatically rejoins the cluster in the **replica** role.

Figu



13.3.2.3. Replica Failure and Recovery

The figure that follows illustrates a sequence of events whereby the replica suffers a failure leaving the master to continue processing alone. Later the replica is repaired and is restarted. It rejoins the cluster so that it is once again ready to take over in the event of master failure.

The behavior of the replica failure case is governed by the `designatedPrimary` configuration item. If set true on the master, the master will continue to operate solo without outside intervention when the replica fails. If false, a third-party must designate the master as primary in order for it to continue solo.

The item numbers in this list apply to the numbered boxes in the figure below. This example assumes that `designatedPrimary` is true on the original master node.

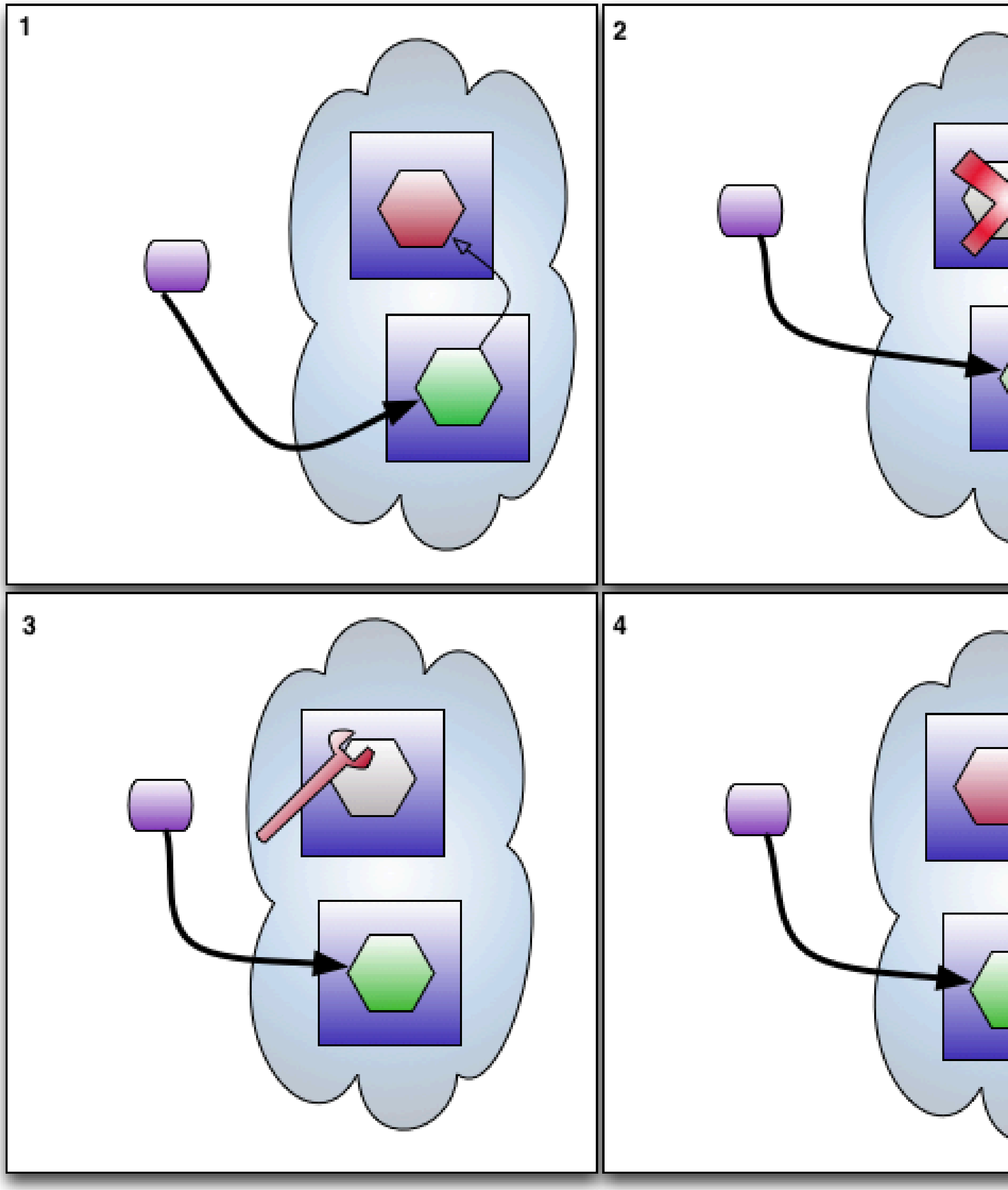
1. System operating normally

2. Replica suffers a failure. Master realises that replica longer in contact but as `designatedPrimary` is true, master continues processing solo and thus client connections are uninterrupted by the loss of the replica. System continues operating normally, albeit with a single node.

3. Replica is repaired.

4. After catching up with missed work, replica is once again ready to take over in the event of master failure.

Figure 13.4. Failure of replica and subsequent recovery sequence



13.3.2.4. Network Partition and Recovery

The figure below illustrates the sequence of events that would occur if the network between master and replica were to suffer a partition, and the nodes were out of contact with one and other.

As with Replica Failure and Recovery, the behaviour is governed by the `designatedPrimary`. Only if `designatedPrimary` is true on the master, will the master continue solo.

The item numbers in this list apply to the numbered boxes in the figure below. This example assumes that `designatedPrimary` is true on the original master node.

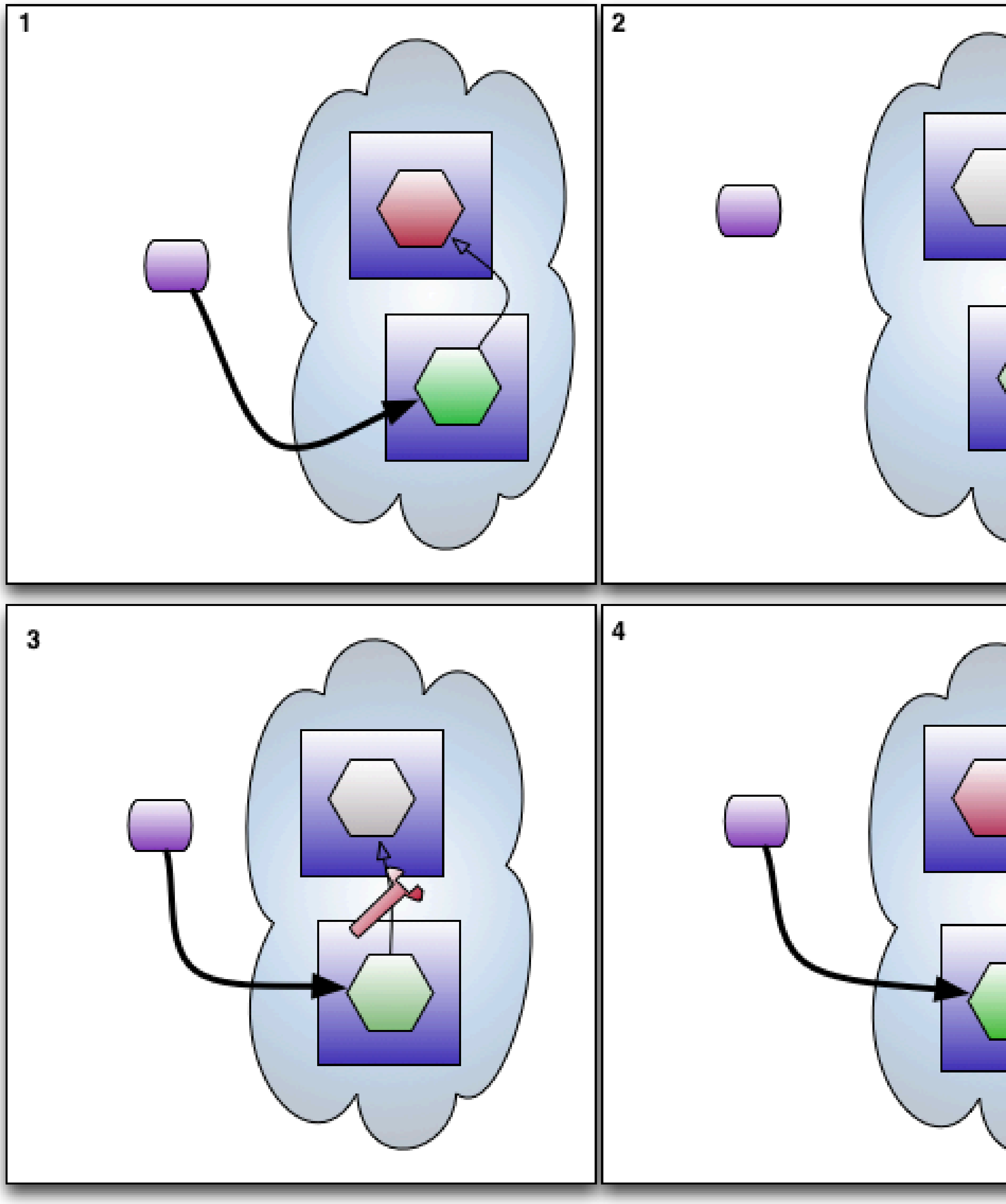
1. System operating normally

2. Network suffers a failure. Master realises that replica longer in contact but as `designatedPrimary` is true, master continues processing solo and thus client connections are uninterrupted by the network partition between master and replica.

3. Network is repaired.

4. After catching up with missed work, replica is once again ready to take over in the event of master failure. System operating normally again.

Figure 13.5. Partition of the network separating master and replica



13.3.2.5. Split Brain

A split-brain [http://en.wikipedia.org/wiki/Split-brain_(computing)] is a situation where the two node cluster has two masters. BDB normally strives to prevent this situation arising by preventing two nodes in a cluster being master at the same time. However, if the network suffers a partition, and the third-party intervenes incorrectly and makes the replica a second master a split-brain will be formed and both masters will proceed to perform work **independently** of one and other.

There is no automatic recovery from a split-brain.

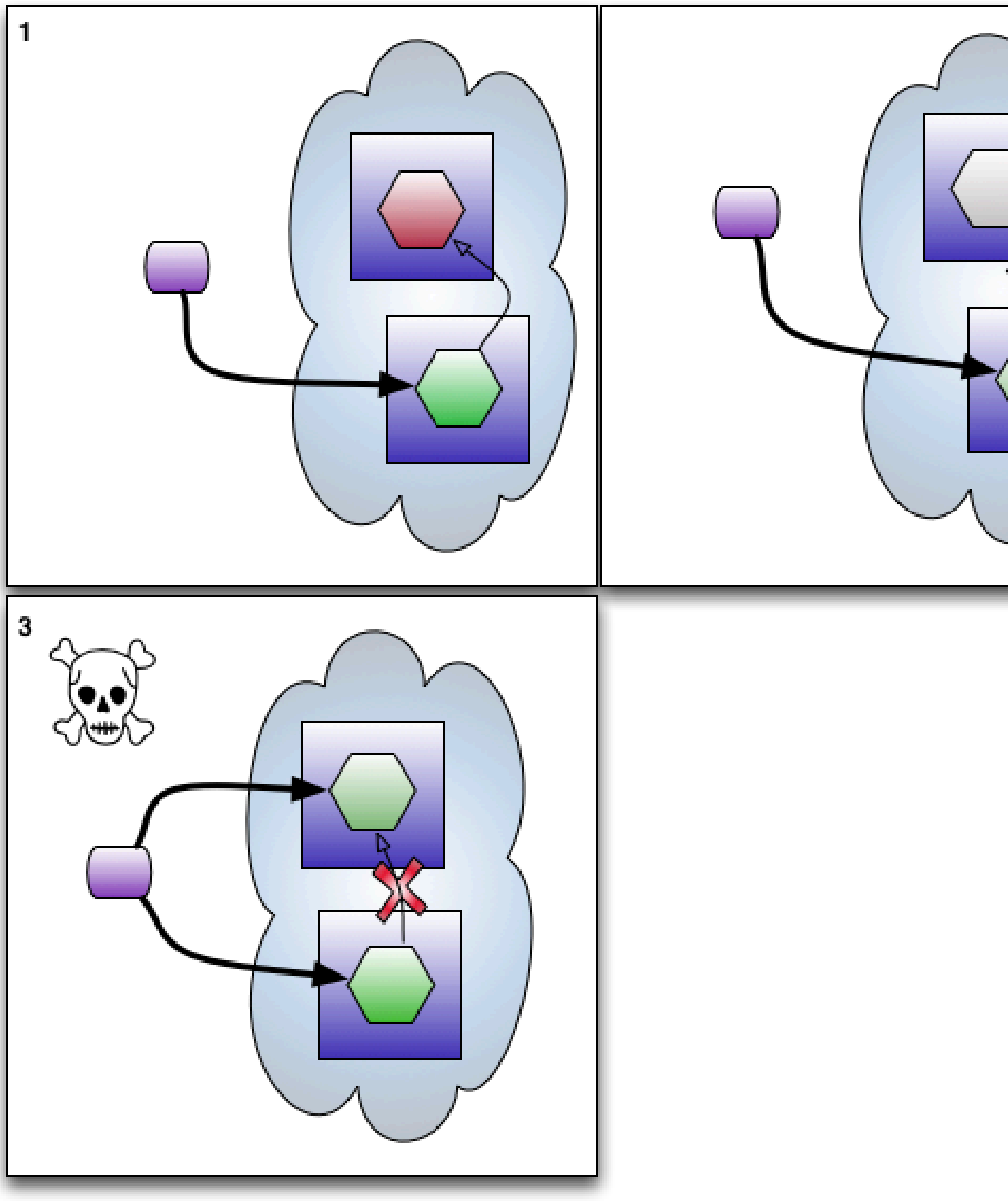
Manual intervention will be required to choose which store will be retained as master and which will be discarded. Manual intervention will be required to identify and repeat the lost business transactions.

The item numbers in this list apply to the numbered boxes in the figure below.

1. System operating normally
2. Network suffers a failure. Master realises that replica longer in contact but as `designatedPrimary` is true, master continues processing solo. Client connections are uninterrupted by the network partition.

   A third-party **erroneously** designates the replica as primary while the original master continues running (now solo).
3. As the nodes cannot see one and other, both behave as masters. Clients may perform work against both master nodes.

Figure 13.6. Split Brain



13.4. Multi Node Cluster

Multi node clusters, that is clusters where the number of nodes is three or more, are not yet ready for use.

13.5. Configuring a Virtual Host to be a node

To configure a virtualhost as a cluster node, configure the virtualhost.xml in the following manner:

Example 13.1. Configuring a VirtualHost to use the BDBHAMessageStore

```
<virtualhosts>
  <virtualhost>
    <name>vhostname</name>
    <vhostname>
      <store>
 <class>org.apache.qpid.server.store.berkeleydb.BDBHAMessageStore</class>
 <environment-path>${QPID_WORK}/bdbhastore/vhostname</environment-path>
 <highAvailability>
   <groupName>myclustername</groupName>
   <nodeName>mynode1</nodeName>
   <nodeHostPort>node1host:port</nodeHostPort>
   <helperHostPort>node1host:port</helperHostPort>
   <durability>NO_SYNC\,NO_SYNC\,SIMPLE_MAJORITY</durability>
   <coalescingSync>true|false</coalescingSync>
   <designatedPrimary>true|false</designatedPrimary>
 </highAvailability>
      </store>
      ...
   </vhostname>
  </virtualhost>
</virtualhosts>
```

The `groupName` is the name of logical name of the cluster. All nodes within the cluster must use the same `groupName` in order to be considered part of the cluster.

The `nodeName` is the logical name of the node. All nodes within the cluster must have a unique name. It is recommended that the node name should be chosen from a different nomenclature from that of the servers on which they are hosted, in case the need arises to move node to a new server in the future.

The `nodeHostPort` is the hostname and port number used by this node to communicate with the the other nodes in the cluster. For the hostname, an IP address, hostname or fully qualified hostname may be used. For the port number, any free port can be used. It is important that this address is stable over time, as BDB records and uses this address internally.

The `helperHostPort` is the hostname and port number that new nodes use to discover other nodes within the cluster when they are newly introduced to the cluster. When configuring the first node, set the `helperHostPort` to its own `nodeHostPort`. For the second and subsequent nodes, set their `helperHostPort` to that of the first node.

`durability` controls the durability guarantees made by the cluster. It is important that all nodes use the same value for this property. The default value is NO_SYNC\,NO_SYNC\,SIMPLE_MAJORITY. Owing to the internal use of Apache Commons Config, it is currently necessary to escape the commas within the durability string.

`coalescingSync` controls the coalescing-sync mode of Qpid. It is important that all nodes use the same value. If omitted, it defaults to true.

The `designatedPrimary` is applicable only to the two-node case. It governs the behaviour of a node when the other node fails or becomes uncontactable. If true, the node will be designated as primary at startup and will be able to continue operating as a single node master. If false, the node will transition to an unavailable state until a third-party manually designates the node as primary or the other node is restored. It is suggested that the node that normally fulfils the role of master is set true in config file and the node that is normally replica is set false. Be aware that setting both nodes to true will lead to a failure to start up, as both cannot be designated at the point of contact. Designating both nodes as primary at runtime (using the JMX interface) will lead to a split-brain in the case of network partition and must be avoided.

Note

Usage of domain names in `helperHostPort` and `nodeHostPort` is more preferebale over IP addresses due to the tendency of more frequent changes of the last over the former. If server IP address changes but domain name remains the same the HA cluster can continue working as normal in case when domain names are used in cluster configuration. In case when IP addresses are used and they are changed with the time than Qpid JMX API for HA can be used to change the addresses or remove the nodes from the cluster.

13.5.1. Passing BDB environment and replication configuration options

It is possible to pass BDB environment [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/EnvironmentConfig.html] and replication [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/rep/ReplicationConfig.html] configuration options from the virtualhost.xml.

Environment configuration options are passed as described in Section 14.4.1, “Passing BDB environment configuration options”

Replication configuration option are passed using `repConfig` elements with the `store` element.

For example, to override the BDB replication configuration option `je.rep.electionsPrimaryRetries`.

```
        ...
      </highAvailability>
      ...
      <repConfig>
        <name>je.rep.electionsPrimaryRetries</name>
        <value>3</value>
      </repConfig>
      ...
    </store>
```

13.6. Durability Guarantees

The term durability [http://en.wikipedia.org/wiki/ACID#Durability] is used to mean that once a transaction is committed, it remains committed regardless of subsequent failures. A highly durable system is one where loss of a committed transaction is extermely unlikely, whereas with a less durable system loss of a transaction is likely in a greater number of scenarios. Typically, the more highly durable a system the slower and more costly it will be.

Qpid exposes the all the durability controls [http://oracle.com/cd/E17277_02/html/ReplicationGuide/txn-management.html#durabilitycontrols] offered by by BDB JE JA and a Qpid specific optimisation called **coalescing-sync** which defaults to enabled.

13.6.1. BDB Durability Controls

BDB expresses durability as a triplet with the following form:

```
<master sync policy>,<replica sync policy>,<replica acknowledgement policy>
```

The sync polices controls whether the thread performing the committing thread awaits the successful completion of the write, or the write and sync before continuing. The master sync policy and replica sync policy need not be the same.

For master and replic sync policies, the available values are: SYNC [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/Durability.SyncPolicy.html#SYNC], WRITE_NO_SYNC [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/Durability.SyncPolicy.html#WRITE_NO_SYNC], NO_SYNC [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/Durability.SyncPolicy.html#NO_SYNC]. SYNC is offers the highest durability whereas NO_SYNC the lowest.

Note: the combination of a master sync policy of SYNC and coalescing-sync true would result in poor performance with no corresponding increase in durability guarantee. It cannot not be used.

The acknowledgement policy defines whether when a master commits a transaction, it also awaits for the replica(s) to commit the same transaction before continuing. For the two-node case, ALL and SIMPLE_MAJORITY are equal.

For acknowledgement policy, the available value are: ALL [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/Durability.ReplicaAckPolicy.html#ALL], SIMPLE_MAJORITY [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/Durability.ReplicaAckPolicy.html#SIMPLE_MAJORITY] NONE [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/Durability.ReplicaAckPolicy.html#NONE].

13.6.2. Coalescing-sync

If enabled (the default) Qpid works to reduce the number of separate file-system sync [http://docs.oracle.com/javase/6/docs/api/java/io/FileDescriptor.html#sync()] operations performed by the **master** on the underlying storage device thus improving performance. It does this coalescing separate sync operations arising from the different client commits operations occuring at approximately the same time. It does this in such a manner not to reduce the ACID guarantees of the system.

Coalescing-sync has no effect on the behaviour of the replicas.

13.6.3. Default

The default durability guarantee is `NO_SYNC, NO_SYNC, SIMPLE_MAJORITY` with coalescing-sync enabled. The effect of this combination is described in the table below. It offers a good compromise between durability guarantee and performance with writes being guaranteed on the master and the additional guarantee that a majority of replicas have received the transaction.

13.6.4. Examples

Here are some examples illustrating the effects of the durability and coalescing-sync settings.

Table 13.1. Effect of different durability guarantees

	Durability	Coalescing-sync	Description
1	NO_SYNC, NO_SYNC, SIMPLE_MAJORITY	true	Before the commit returns to the client, the transaction will be written/sync'd to the Master's disk (effect of coalescing-sync) and a majority of the replica(s) will have acknowledged the **receipt** of the transaction. The replicas will write and sync the transaction to their disk at a point in the future governed by ReplicationMutableConfig#LOG_FLUSH_ [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/rep/ReplicationMutableConfig.html#LOG_FLU
2	NO_SYNC, WRITE_NO_SYNC, SIMPLE_MAJORITY	true	Before the commit returns to the client, the transaction will be written/sync'd to the Master's disk (effect of coalescing-sync and a majority of the replica(s) will have acknowledged the **write** of the transaction to their disk. The replicas will sync the transaction to disk at a point in the future with an upper bound governed by ReplicationMutableConfig#LOG_FLUSH_
3	NO_SYNC, NO_SYNC, NONE	false	After the commit returns to the client, the transaction is neither guaranteed to be written to the disk of the master nor received by any of the replicas. The master and replicas will write and sync the

	Durability	Coalescing-sync	Description
			transaction to their disk at a point in the future with an upper bound governed by ReplicationMutableConfig#LOG_FLUSH_ This offers the weakest durability guarantee.

13.7. Client failover configuration

The details about format of Qpid connection URLs can be found at section Connection URLs [../../Programming-In-Apache-Qpid/html/QpidJNDI.html#section-jms-connection-url] within the client documentation.

The failover policy option in the connection URL for the HA Cluster should be set to *roundrobin*. The Master broker should be put into a first place in *brokerlist* URL option. The recommended value for *connectdelay* option in broker URL should be set to the value greater than 1000 milliseconds. If it is desired that clients re-connect automatically after a master to replica failure, `cyclecount` should be tuned so that the retry period is longer than the expected length of time to perform the failover.

Example 13.2. Example of connection URL for the HA Cluster

```
amqp://guest:guest@clientid/test?brokerlist='tcp://localhost:5672?connectdelay='2000'&retries='3';tcp://
localhost:5671?connectdelay='2000'&retries='3';tcp://localhost:5673?
connectdelay='2000'&retries='3''&failover='roundrobin?cyclecount='30''
```

13.8. Qpid JMX API for HA

Qpid exposes the BDB HA store information via its JMX interface and provides APIs to remove a Node from the group, update a Node IP address, and assign a Node as the designated primary.

An instance of the `BDBHAMessageStore` MBean is instantiated by the broker for the each virtualhost using the HA store.

The reference to this MBean can be obtained via JMX API using an ObjectName like *org.apache.qpid:type=BDBHAMessageStore,name="<virtualhost name>"* where <virtualhost name> is the name of a specific virtualhost on the broker.

Table 13.2. Mbean `BDBHAMessageStore` attributes

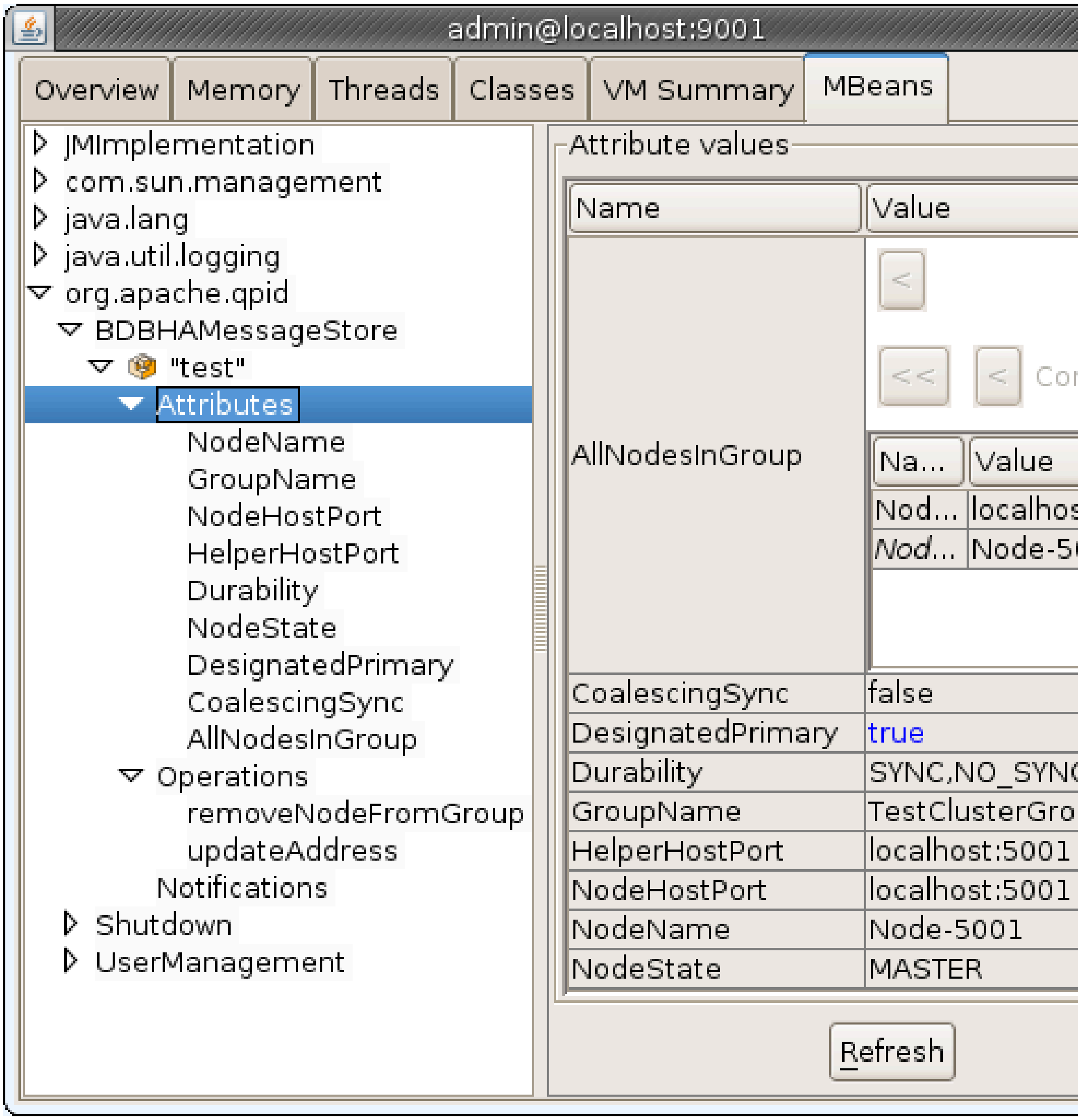
Name	Type	Accessibility	Description
GroupName	String	Read only	Name identifying the group
NodeName	String	Read only	Unique name identifying the node within the group
NodeHostPort	String	Read only	Host/port used to replicate data between this node and others in the group

Name	**Type**	**Accessibility**	**Description**
HelperHostPort	String	Read only	Host/port used to allow a new node to discover other group members
NodeState	String	Read only	Current state of the node
ReplicationPolicy	String	Read only	Node replication durability
DesignatedPrimary	boolean	Read/Write	Designated primary flag. Applicable to the two node case.
CoalescingSync	boolean	Read only	Coalescing sync flag. Applicable to the master sync policies NO_SYNC and WRITE_NO_SYNC only.
getAllNodesInGroup	TabularData	Read only	Get all nodes within the group, regardless of whether currently attached or not

Table 13.3. Mbean `BDBHAMessageStore` operations

Operation	**Parameters**	**Returns**	**Description**
removeNodeFromGroup	*nodeName*, name of node, string	void	Remove an existing node from the group
updateAddress	• *nodeName*, name of node, string • *newHostName*, new host name, string • *newPort*, new port number, int	void	Update the address of another node. The node must be in a STOPPED state.

Figure 13.7. BDBHAMessageStore view from jconsole.



Name	Value
AllNodesInGroup	Na... / Value; Nod... / localhos; Nod... / Node-50
CoalescingSync	false
DesignatedPrimary	true
Durability	SYNC,NO_SYNC
GroupName	TestClusterGro
HelperHostPort	localhost:5001
NodeHostPort	localhost:5001
NodeName	Node-5001
NodeState	MASTER

Example 13.3. Example of java code to get the node state value

```
Map<String, Object> environment = new HashMap<String, Object>();

// credentials: user name and password
environment.put(JMXConnector.CREDENTIALS, new String[] {"admin","admin"});
JMXServiceURL url =  new JMXServiceURL("service:jmx:rmi:///jndi/rmi://localhost:90
JMXConnector jmxConnector = JMXConnectorFactory.connect(url, environment);
MBeanServerConnection mbsc =  jmxConnector.getMBeanServerConnection();

ObjectName queueObjectName = new ObjectName("org.apache.qpid:type=BDBHAMessageStor
String state = (String)mbsc.getAttribute(queueObjectName, "NodeState");

System.out.println("Node state:" + state);
```

Example system output:

```
Node state:MASTER
```

13.9. Monitoring cluster

In order to discover potential issues with HA Cluster early, all nodes in the Cluster should be monitored on regular basis using the following techniques:

- Broker log files scrapping for WARN or ERROR entries and operational log entries like:
  - *MST-1007 :* Store Passivated. It can indicate that Master virtual host has gone down.
  - *MST-1006 :* Recovery Complete. It can indicate that a former Replica virtual host is up and became the Master.
- Disk space usage and system load using system tools.
- Berkeley HA node status using `DbPing` [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/rep/util/DbPing.html] utility.

  Example 13.4. Using `DbPing` utility for monitoring HA nodes.

  java -jar je-5.0.97.jar DbPing -groupName TestClusterGroup -nodeName Node-5001 -nodeHost localhost:5001 -socketTimeout 10000

  ```
  Current state of node: Node-5001 from group: TestClusterGroup
    Current state: MASTER
    Current master: Node-5001
    Current JE version: 5.0.97
    Current log version: 8
    Current transaction end (abort or commit) VLSN: 165
    Current master transaction end (abort or commit) VLSN: 0
    Current active feeders on node: 0
    Current system load average: 0.35
  ```

  In the example above `DbPing` utility requested status of Cluster node with name *Node-5001* from replication group *TestClusterGroup* running on host *localhost:5001*. The state of the node was reported into a system output.

- Using Qpid broker JMX interfaces.

  Mbean `BDBHAMessageStore` can be used to request the following node information:

  - *NodeState* indicates whether node is a Master or Replica.

- *Durability* replication durability.
- *DesignatedPrimary* indicates whether Master node is designated primary.
- *GroupName* replication group name.
- *NodeName* node name.
- *NodeHostPort* node host and port.
- *HelperHostPort* helper host and port.
- *AllNodesInGroup* lists of all nodes in the replication group including their names, hosts and ports.

For more details about `BDBHAMessageStore` MBean please refer section Qpid JMX API for HA

13.10. Disk space requirements

Disk space is a critical resource for the HA Qpid broker.

In case when a Replica goes down (or falls behind the Master in 2 node cluster where the Master is designated primary) and the Master continues running, the non-replicated store files are kept on the Masters disk for the period of time as specified in *je.rep.repStreamTimeout* JE setting in order to replicate this data later when the Replica is back. This setting is set to 1 hour by default by the broker. The setting can be overridden as described in Section 13.5.1, “Passing BDB environment and replication configuration options”.

Depending from the application publishing/consuming rates and message sizes, the disk space might become overfull during this period of time due to preserved logs. Please, make sure to allocate enough space on your disk to avoid this from happening.

13.11. Network Requirements

The HA Cluster performance depends on the network bandwidth, its use by existing traffic, and quality of service.

In order to achieve the best performance it is recommended to use a separate network infrastructure for the Qpid HA Nodes which might include installation of dedicated network hardware on Broker hosts, assigning a higher priority to replication ports, installing a cluster in a separate network not impacted by any other traffic.

13.12. Security

At the moment Berkeley replication API supports only TCP/IP protocol to transfer replication data between Master and Replicas.

As result, the replicated data is unprotected and can be intercepted by anyone having access to the replication network.

Also, anyone who can access to this network can introduce a new node and therefore receive a copy of the data.

In order to reduce the security risks the entire HA cluster is recommended to run in a separate network protected from general access.

13.13. Backups

In order to protect the entire cluster from some cataclysms which might destroy all cluster nodes, backups of the Master store should be taken on a regular basis.

Qpid Broker distribution includes the "hot" backup utility *backup.sh* which can be found at broker bin folder. This utility can perform the backup when broker is running.

backup.sh script invokes `org.apache.qpid.server.store.berkeleydb.BDBBackup` to do the job.

You can also run this class from command line like in an example below:

Example 13.5. Performing store backup by using `BDBBackup` class directly

java -cp qpid-bdbstore-0.28.jar org.apache.qpid.server.store.berkeleydb.BDBBackup -fromdir path/to/store/folder -todir path/to/backup/folder

In the example above BDBBackup utility is called from qpid-bdbstore-0.28.jar to backup the store at *path/to/store/folder* and copy store logs into *path/to/backup/folder*.

Linux and Unix users can take advantage of *backup.sh* bash script by running this script in a similar way.

Example 13.6. Performing store backup by using `backup.sh` bash script

backup.sh -fromdir path/to/store/folder -todir path/to/backup/folder

Note

Do not forget to ensure that the Master store is being backed up, in the event the Node elected Master changes during the lifecycle of the cluster.

13.14. Migration of a non-HA store to HA

Non HA stores starting from schema version 4 (0.14 Qpid release) can be automatically converted into HA store on broker startup if replication is first enabled with the `DbEnableReplication` [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/rep/util/DbEnableReplication.html] utility from the BDB JE jar.

DbEnableReplication converts a non HA store into an HA store and can be used as follows:

Example 13.7. Enabling replication

java -jar je-5.0.97.jar DbEnableReplication -h /path/to/store -groupName MyReplicationGroup -nodeName MyNode1 -nodeHostPort localhost:5001

In the examples above, je jar of version 5.0.97 is used to convert store at */path/to/store* into HA store having replication group name *MyReplicationGroup*, node name *MyNode1* and running on host *localhost* and port *5001*.

After running DbEnableReplication and updating the virtual host store to configuration to be an HA message store, like in example below, on broker start up the store schema will be upgraded to the most recent version and the broker can be used as normal.

Example 13.8. Example of XML configuration for HA message store

```
<store>
    <class>org.apache.qpid.server.store.berkeleydb.BDBHAMessageStore</class>
    <environment-path>/path/to/store</environment-path>
    <highAvailability>
        <groupName>MyReplicationGroup</groupName>
        <nodeName>MyNode1</nodeName>
        <nodeHostPort>localhost:5001</nodeHostPort>
        <helperHostPort>localhost:5001</helperHostPort>
    </highAvailability>
</store>
```

The Replica nodes can be started with empty stores. The data will be automatically copied from Master to Replica on Replica start-up. This will take a period of time determined by the size of the Masters store and the network bandwidth between the nodes.

Note

Due to existing caveats in Berkeley JE with copying of data from Master into Replica it is recommended to restart the Master node after store schema upgrade is finished before starting the Replica nodes.

13.15. Disaster Recovery

This section describes the steps required to restore HA broker cluster from backup.

The detailed instructions how to perform backup on replicated environment can be found here.

At this point we assume that backups are collected on regular basis from Master node.

Replication configuration of a cluster is stored internally in HA message store. This information includes IP addresses of the nodes. In case when HA message store needs to be restored on a different host with a different IP address the cluster replication configuration should be reseted in this case

Oracle provides a command line utility `DbResetRepGroup` [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/rep/util/DbResetRepGroup.html] to reset the members of a replication group and replace the group with a new group consisting of a single new member as described by the arguments supplied to the utility

Cluster can be restored with the following steps:

- Copy log files into the store folder from backup
- Use `DbResetRepGroup` to reset an existing environment. See an example below

  Example 13.9. Reseting of replication group with `DbResetRepGroup`

  java -cp je-5.0.97.jar com.sleepycat.je.rep.util.DbResetRepGroup -h ha-work/Node-5001/bdbstore -groupName TestClusterGroup -nodeName Node-5001 -nodeHostPort localhost:5001

  In the example above `DbResetRepGroup` utility from Berkeley JE of version 5.0.97 is used to reset the store at location *ha-work/Node-5001/bdbstore* and set a replication group to *TestClusterGroup* having a node *Node-5001* which runs at *localhost:5001*.

- Start a broker with HA store configured as specified on running of `DbResetRepGroup` utility.
- Start replica nodes having the same replication group and a helper host port pointing to a new master. The store content will be copied into Replicas from Master on their start up.

13.16. Performance

The aim of this section is not to provide exact performance metrics relating to HA, as this depends heavily on the test environment, but rather showing an impact of HA on Qpid broker performance in comparison with the Non HA case.

For testing of impact of HA on a broker performance a special test script was written using Qpid performance test framework. The script opened a number of connections to the Qpid broker, created producers and consumers on separate connections, and published test messages with concurrent producers into a test queue and consumed them with concurrent consumers. The table below shows the number of producers/consumers used in the tests. The overall throughput was collected for each configuration.

Table 13.4. Number of producers/consumers in performance tests

Test	Number of producers	Number of consumers
1	1	1
2	2	2
3	4	4
4	8	8
5	16	16
6	32	32
7	64	64

The test was run against the following Qpid Broker configurations

- Non HA Broker
- HA 2 Nodes Cluster with durability *SYNC,SYNC,ALL*
- HA 2 Nodes Cluster with durability *WRITE_NO_SYNC,WRITE_NO_SYNC,ALL*
- HA 2 Nodes Cluster with durability *WRITE_NO_SYNC,WRITE_NO_SYNC,ALL* and *coalescing-sync* Qpid mode
- HA 2 Nodes Cluster with durability *WRITE_NO_SYNC,NO_SYNC,ALL* and *coalescing-sync* Qpid mode
- HA 2 Nodes Cluster with durability *NO_SYNC,NO_SYNC,ALL* and *coalescing-sync* Qpid option

The evironment used in testing consisted of 2 servers with 4 CPU cores (2x Intel(r) Xeon(R) CPU 5150@2.66GHz), 4GB of RAM and running under OS Red Hat Enterprise Linux AS release 4 (Nahant Update 4). Network bandwidth was 1Gbit.

We ran Master node on the first server and Replica and clients(both consumers and producers) on the second server.

In non-HA case Qpid Broker was run on a first server and clients were run on a second server.

The table below contains the test results we measured on this environment for different Broker configurations.

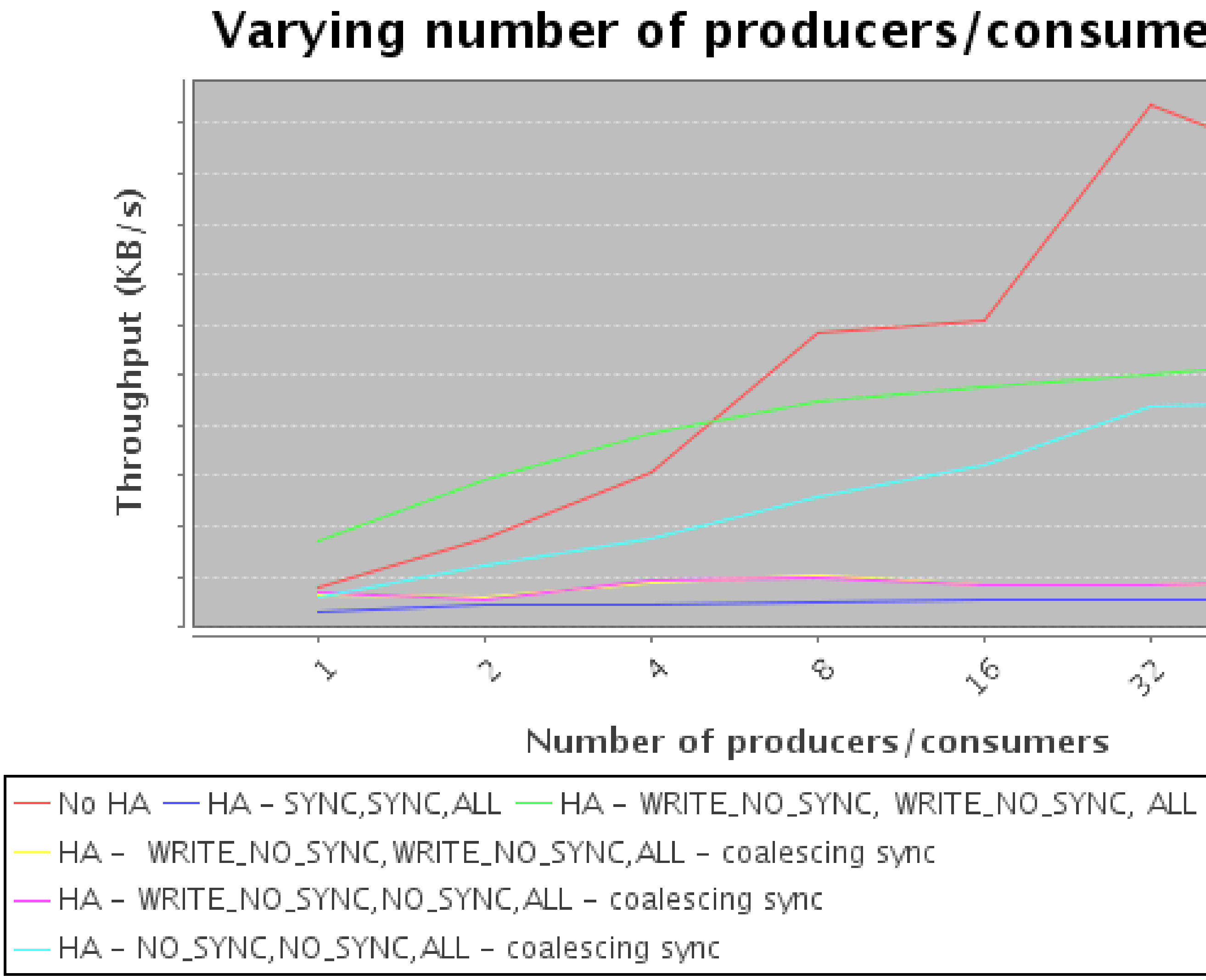
Each result is represented by throughput value in KB/second and difference in % between HA configuration and non HA case for the same number of clients.

Table 13.5. Performance Comparison

Test/Broker	**No HA**	**SYNC, SYNC, ALL**	**WRITE_NO_SYNC, WRITE_NO_SYNC, ALL**	**WRITE_NO_SYNC, WRITE_NO_SYNC, ALL - coalescing-sync**	**WRITE_NO_SYNC, NO_SYNC,ALL - coalescing-sync**	**NO_SYNC, NO_SYNC, ALL - coalescing-sync**
1 (1/1)	0.0%	-61.4%	117.0%	-16.02%	-9.58%	-25.47%
2 (2/2)	0.0%	-75.43%	67.87%	-66.6%	-69.02%	-30.43%
3 (4/4)	0.0%	-84.89%	24.19%	-71.02%	-69.37%	-43.67%
4 (8/8)	0.0%	-91.17%	-22.97%	-82.32%	-83.42%	-55.5%
5 (16/16)	0.0%	-91.16%	-21.42%	-86.6%	-86.37%	-46.99%
6 (32/32)	0.0%	-94.83%	-51.51%	-92.15%	-92.02%	-57.59%
7 (64/64)	0.0%	-94.2%	-41.84%	-89.55%	-89.55%	-50.54%

The figure below depicts the graphs for the performance test results

Figure 13.8. Test results



On using durability *SYNC,SYNC,ALL* (without coalescing-sync) the performance drops significantly (by 62-95%) in comparison with non HA broker.

Whilst, on using durability *WRITE_NO_SYNC,WRITE_NO_SYNC,ALL* (without coalescing-sync) the performance drops by only half, but with loss of durability guarantee, so is not recommended.

In order to have better performance with HA, Qpid Broker comes up with the special mode called coalescing-sync, With this mode enabled, Qpid broker batches the concurrent transaction commits and syncs transaction data into Master disk in one go. As result, the HA performance only drops by 25-60% for durability *NO_SYNC,NO_SYNC,ALL* and by 10-90% for *WRITE_NO_SYNC,WRITE_NO_SYNC,ALL*.

Chapter 14. Virtual Host XML configuration file

14.1. Introduction

This chapter describes how to configure Virtual Hosts using an XML file.

This is no longer the preferred approach for defining a VirtualHost and will likely be removed in a future release, however it is currently neccessary to support certain use cases such as per-virtualhost attribute configuration, and specialised store configuration such as for the BDB HA Message Store

Each XML Virtual Host configuration file can hold configuration for a single Virtual Host or multiple Virtual Hosts. For an example file (with multiple VirtualHosts), see Section 14.15, “An example of virtual host configuration file”

14.2. Configuring ACL

Section 11.3, “Access Control Lists” provides the details of ACL, rules, formats, etc.

To apply an ACL on a single virtualhost named *test*, add the following to the virtualhosts.xml:

```
<virtualhost>
...
    <name>test</name>
    <test>
      ...
      <security>                               1
        ...
        <acl>${conf}/vhost_test.acl</acl> 2
        ...
      </security>
      ...
    </test>
</virtualhost>
```

1 A security section of configuration is used to declare the ACL
2 A path to an ACL file is configured (assuming that *conf* has been set to a suitable location such as ${QPID_HOME}/etc)

14.3. Configuring MemoryMessageStore

An example of MemoryMessageStore configuration for a virtual host is shown below:

Example 14.1. Configuring a VirtualHost to use the MemoryMessageStore

```
<virtualhosts>
  <virtualhost>
```

```
    <name>vhostname</name>
    <vhostname>
      <store>
        <class>org.apache.qpid.server.store.MemoryMessageStore</class
      </store>
      ...
    </vhostname>
  </virtualhost>
</virtualhosts>
```

14.4. Configuring BDBMessageStore

In order to use the BDBMessageStore, you must configure it for each VirtualHost desired by updating the store element to specify the associated store class and provide a directory location for the data to be written, as shown below.

Example 14.2. Configuring a VirtualHost to use the BDBMessageStore

```
<virtualhosts>
  <virtualhost>
    <name>vhostname</name>
    <vhostname>
      <store>
        <class>org.apache.qpid.server.store.berkeleydb.BDBMessageStore</class>
        <environment-path>${QPID_WORK}/bdbstore/vhostname</environment-path>
      </store>
      ...
    </vhostname>
  </virtualhost>
</virtualhosts>
```

14.4.1. Passing BDB environment configuration options

It is possible to pass BDB environment [http://docs.oracle.com/cd/E17277_02/html/java/com/sleepycat/je/EnvironmentConfig.html] from the virtualhost.xml. Environment configuration options are passed using `envConfig` elements within the `store` element.

For example, to override the BDB environment configuration options `je.cleaner.threads` and `je.log.fileMax`

Example 14.3. Configuring BDB Environment Configuration

```
      <store>
 ...
 <envConfig>
   <name>je.cleaner.threads</name>
   <value>2</value>
 </envConfig>
 <envConfig>
```

```
    <name>je.log.fileMax</name>
    <value>5000000</value>
  </envConfig>
    ...
       </store>
```

14.5. Configuring BDBHAMessageStore

See Section 13.5, “Configuring a Virtual Host to be a node”

14.6. Configuring DerbyMessageStore

In order to use the DerbyMessageStore, you must configure it for each VirtualHost desired by updating the store element to specify the associated store class and provide a directory location for the data to be written, as shown below.

Example 14.4. Configuring a VirtualHost to use the DerbyMessageStore

```
<virtualhosts>
  <virtualhost>
    <name>vhostname</name>
    <vhostname>
      <store>
        <class>org.apache.qpid.server.store.DerbyMessageStore</class>
        <environment-path>${QPID_WORK}/derbystore/vhostname</environment-path>
      </store>
      ...
    </vhostname>
  </virtualhost>
</virtualhosts>
```

14.7. Configuring JDBCMessageStore

JDBCMessageStore can be configured on VirtualHost as in example shown below:

Example 14.5. Configuring a VirtualHost to use the JDBCMessageStore

```
<virtualhosts>
  <virtualhost>
    <name>vhostname</name>
    <vhostname>
      <store>
        <class>org.apache.qpid.server.store.jdbc.JDBCMessageStore</class>
        <connectionUrl>jdbc:oracle:thin:guest@guest//localhost:1521/orcl</connecti
      </store>
      ...
    </vhostname>
  </virtualhost>
</virtualhosts>
```

14.8. Configuring Exchanges

To declare Exchanges within Virtual Host configuration, add the appropriate xml to the virtualhost.xml configuration file within the `exchanges` element. An example of such declaration is shown below:

Example 14.6. Configuring Exchanges on VirtualHost

```
<virtualhosts>
  <virtualhost>
    <name>vhostname</name>
      ...
            <exchanges>
                <exchange>
                    <type>direct</type>
                    <name>test.direct</name>
                    <durable>true</durable>
                </exchange>
                <exchange>
                    <type>topic</type>
                    <name>test.topic</name>
                </exchange>
            </exchanges>
      ...
    </vhostname>
  </virtualhost>
</virtualhosts>
```

14.9. Configuring Queues

To create a priority, sorted or LVQ queue within configuration, add the appropriate xml to the virtualhost.xml configuration file within the `queues` element.

14.9.1. Simple

For declaration of a simple queue define a queue entry in the virtual host configuration as in example below

Example 14.7. Configuring a simple queue

```
<queue>
    <name>my-simple-queue</name>
    <my-simple-queue>
        <exchange>amq.direct</exchange>
        <durable>true</durable>
    </my-simple-queue>
</queue>
```

14.9.2. Priority

To defining a priority queue, add a <priority>true</priority> element. By default the queue will have 10 distinct priorities.

Example 14.8. Configuring a priority queue

```
<queue>
    <name>myqueue</name>
    <myqueue>
        <exchange>amq.direct</exchange>
        <priority>true</priority>
    </myqueue>
</queue>
```

If you require fewer priorities, it is possible to specify a `priorities` element (whose value is a integer value between 2 and 10 inclusive) which will give the queue that number of distinct priorities. When messages are sent to that queue, their effective priority will be calculated by partitioning the priority space. If the number of effective priorities is 2, then messages with priority 0-4 are treated the same as "lower priority" and messages with priority 5-9 are treated equivalently as "higher priority".

Example 14.9. Configuring a priority queue with fewer priorities

```
<queue>
    <name>myqueue</name>
    <myqueue>
        <exchange>amq.direct</exchange>
        <priority>true</priority>
        <priorities>4</priorities>
    </myqueue>
</queue>
```

14.9.3. Sorted

To define a sorted queue, add a `sortKey` element. The value of the `sortKey` element defines the message property to use the value of when sorting the messages put onto the queue.

Example 14.10. Configuring a sorted queue

```
<queue>
    <name>myqueue</name>
    <myqueue>
        <exchange>amq.direct</exchange>
        <sortKey>message-property-to-sort-by</sortKey>
    </myqueue>
</queue>
```

14.9.4. LVQ

To define a LVQ, add a `lvq` element with the value `true`. Without any further configuration this will define an LVQ which uses the JMS message property `qpid.LVQ_key` as the key for replacement.

Example 14.11. Configuring a LVQ queue

```
<queue>
    <name>myqueue</name>
    <myqueue>
        <exchange>amq.direct</exchange>
        <lvq>true</lvq>
    </myqueue>
</queue>
```

If you wish to define your own property then you can do so using the `lvqKey` element.

Example 14.12. Configuring a LVQ queue with custom message property name

```
<queue>
    <name>myqueue</name>
    <myqueue>
        <exchange>amq.direct</exchange>
        <lvq>true</lvq>
        <lvqKey>ISIN</lvqKey>
    </myqueue>
</queue>
```

14.9.5. Configuring queue with arguments

Queue arguments can be configured in virtual host configuration file. An element *argument* is used to specify a queue argument as a name-value pair separated with equal character. Any number of queue arguments can be provided for a queue in their own *argument* elements.

The following example demonstrates how to configure queue with two arguments.

Example 14.13. Setting arbitrary queue arguments

```

<queue>
    <name>myQueue</name>
    <myQueue>
        <argument>qpid.group_header_key=JMSXgroupID</argument>
        <argument>qpid.shared_msg_group=1</argument>
    </myQueue>
</queue>
```

14.10. Queue Binding

A queue can be bound to an exchange in virtual host configuration file by providing an exchange name in element *exchange* within the queue configuration element having the same name as a queue. If exchange element is omitted in queue configuration then such queue is bound to a default exchange only. With configuration file it is only possible to bind queue to a single exchange.

An element *routingKey* is used to specify a custom binding key. It is an optional element, and, if it is not set then queue is bound to an exchange with a binding key equals to a queue name. Any number of binding keys can be configured.

Binding arguments can be set with each binding key. An element *bindingArgument* is used to specify a binding argument as a name-value pair separated with equal character. Any number of binding arguments can be provided for a binding key in their own *bindingArgument* elements. All of them should be contained within an element having the same name as a binding key.

The following example demonstrates how to bind queue *testQueue* to a default topic exchange using routing key *testRoutingKey* and binding arguments for message selector and no local.

Example 14.14. Queue Binding Example

```
<queue>
    <name>testQueue</name>
    <testQueue>
        <exchange>amq.topic</exchange>
        <routingKey>testRoutingKey</routingKey>
        <testRoutingKey>
            <bindingArgument>x-filter-jms-selector=application='app1'</bindingArgu
            <bindingArgument>x-qpid-no-local=</bindingArgument>
        </testRoutingKey>
    </testQueue>
</queue>
```

14.11. Configuring of Producer Flow Control

Flow control capacity and flow resume capacity are required to set on a queue or virtual host to enable Producer flow control.

Example 14.15. Configuring a queue depth limit

```
<queue>
    <name>test</name>
    <test>
        <exchange>amq.direct</exchange>

        <!-- set the queue capacity to 10Mb -->
        <capacity>10485760</capacity>

        <!-- set the resume capacity to 8Mb -->
        <flowResumeCapacity>8388608</flowResumeCapacity>
    </test>
</queue>
```

The default for all queues on a virtual host can also be set

Example 14.16. Configuring of default queue depth limit on virtualhost

```
<virtualhosts>
    <virtualhost>
        <name>localhost</name>
        <localhost>

            <!-- set the queue capacity to 10Mb -->
            <capacity>10485760</capacity>
            <!-- set the resume capacity to 8Mb -->
            <flowResumeCapacity>8388608</flowResumeCapacity>
        </localhost>
    </virtualhost>
</virtualhosts>
```

14.12. Configuring of Disk Quota-based Flow Control

An example of quota configuration for the BDB message store is provided below.

Example 14.17. Configuring a limit on a store

```
<store>
   <class>org.apache.qpid.server.store.berkeleydb.BDBMessageStore</class>
   <environment-path>${work}/bdbstore/test</environment-path>
   <overfull-size>50000000</overfull-size>
   <underfull-size>45000000</underfull-size>
</store>
```

14.13. Configuring Transaction Timeouts

The JMS transaction timeouts are configured on each virtual host defined in the XML configuration files.

The default values for each of the parameters is 0, indicating that the particular check is disabled.

Any or all of the parameters can be set, using the desired value in milliseconds, and will be checked each time the housekeeping process runs, usually set to run every 30 seconds in standard configuration. The meaning of each property is as follows:

- openWarn - the time a transaction can be open for (with activity occurring on it) after which a warning alert will be issued.
- openClose - the time a transaction can be open for before the connection it is on is closed.
- idleWarn - the time a transaction can be idle for (with no activity occurring on it) after which a warning alert will be issued.
- idleClose - the time a transaction can be idle for before the connection it is on is closed.

The virtualhosts configuration is shown below, and must occur inside the //virtualhosts/virtualhost/name/ elements:

Example 14.18. Configuring producer transaction timeout

```
<transactionTimeout>
```

```
    <openWarn>10000</openWarn>
    <openClose>20000</openClose>
    <idleWarn>5000</idleWarn>
    <idleClose>15000</idleClose>
</transactionTimeout>
```

14.14. Configuring DLQs/Maximum Delivery Count

In the below configuration it can be seen that DLQs/Maximum Delivery Count are enabled at the virtual host "localhost" with maximum delivery count set to 5 and disable for virtual host "dev-only".

As 'dev-only-main-queue' has its own configuration specified, this value overrides all others and causes the features to be enabled for this queue. In contrast to this, 'dev-only-other-queue' does not specify its own value and picks up the false value specified for its parent virtualhost, causing the DLQ/Maximum Delivery Count features to be disabled for this queue. Any such queue in the 'dev-only' virtualhost which does not specify its own configuration value will have the DLQ/Maximum Delivery Count feature disabled.

The queue 'localhost-queue' has the DLQ/Maximum Delivery Count features disabled. Any other queue in the 'localhost' virtualhost which does not specify its own configuration value will have the features enabled (inherited from parent virtual host).

Example 14.19. Enabling DLQs and maximum delivery count at virtualhost and queue level within virtualhosts.xml

```
<virtualhosts>
 ...
 <virtualhost>
  <name>dev-only</name>
  <dev-only>
   <queues>
    <deadLetterQueues>false</deadLetterQueues>
    <maximumDeliveryCount>0</maximumDeliveryCount>
    <queue>
     <name>dev-only-main-queue</name>
     <dev-only-main-queue>
      <deadLetterQueues>true</deadLetterQueues>
      <maximumDeliveryCount>3</maximumDeliveryCount>
     </dev-only-main-queue>
    </queue>
    <queue>
     <name>dev-only-other-queue</name>
    </queue>
   </queues>
  </dev-only>
 </virtualhost>
 <virtualhost>
  <name>localhost</name>
  <localhost>
   <queues>
    <deadLetterQueues>true</deadLetterQueues>
    <maximumDeliveryCount>5</maximumDeliveryCount>
    <queue>
     <name>localhost-queue</name>
     <deadLetterQueues>false</deadLetterQueues>
    </queue>
   </queues>
  </localhost>
 </virtualhost>
 ...
</virtualhosts>
```

14.15. An example of virtual host configuration file

Example 14.20. An example of virtual host configuration file

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<virtualhosts>
    <virtualhost>
        <name>localhost</name>
        <localhost>
            <store>
                <class>org.apache.qpid.server.store.MemoryMessageStore</class>
                <!--<class>org.apache.qpid.server.store.derby.DerbyMessageStore</c
                <environment-path>${QPID_WORK}/derbystore/localhost</environment-p
            </store>

            <housekeeping>
                <threadCount>2</threadCount>
                <checkPeriod>20000</checkPeriod>
            </housekeeping>

            <exchanges>
                <exchange>
                    <type>direct</type>
                    <name>test.direct</name>
                    <durable>true</durable>
                </exchange>
                <exchange>
                    <type>topic</type>
                    <name>test.topic</name>
                </exchange>
            </exchanges>
            <queues>
                <exchange>amq.direct</exchange>
                <maximumQueueDepth>4235264</maximumQueueDepth>
                <!-- 4Mb -->
                <maximumMessageSize>2117632</maximumMessageSize>
                <!-- 2Mb -->
                <maximumMessageAge>600000</maximumMessageAge>
                <!-- 10 mins -->
                <maximumMessageCount>50</maximumMessageCount>
                <!-- 50 messages -->

                <queue>
                    <name>queue</name>
                </queue>
                <queue>
                    <name>ping</name>
                </queue>
                <queue>
                    <name>test-queue</name>
                    <test-queue>
                        <exchange>test.direct</exchange>
                        <durable>true</durable>
                    </test-queue>
                </queue>
                <queue>
                    <name>test-ping</name>
                    <test-ping>
                        <exchange>test.direct</exchange>
                    </test-ping>
                </queue>

            </queues>
        </localhost>
    </virtualhost>

    <virtualhost>
        <name>development</name>
        <development>
            <store>
                <class>org.apache.qpid.server.store.MemoryMessageStore</class>
                <!--<class>org.apache.qpid.server.store.derby.DerbyMessageStore</c
                <environment-path>${QPID_WORK}/derbystore/development</environment
            </store>

            <queues>
                <minimumAlertRepeatGap>30000</minimumAlertRepeatGap>
                <maximumMessageCount>50</maximumMessageCount>
                <queue>
                    <name>queue</name>
                    <queue>
                        <exchange>amq.direct</exchange>
                        <maximumQueueDepth>4235264</maximumQueueDepth>
                        <!-- 4Mb -->
                        <maximumMessageSize>2117632</maximumMessageSize>
                        <!-- 2Mb -->
                        <maximumMessageAge>600000</maximumMessageAge>
                        <!-- 10 mins -->
                    </queue>
                </queue>
                <queue>
                    <name>ping</name>
                    <ping>
                        <exchange>amq.direct</exchange>
                        <maximumQueueDepth>4235264</maximumQueueDepth>
                        <!-- 4Mb -->
                        <maximumMessageSize>2117632</maximumMessageSize>
                        <!-- 2Mb -->
                        <maximumMessageAge>600000</maximumMessageAge>
                        <!-- 10 mins -->
                    </ping>
                </queue>
            </queues>
        </development>
    </virtualhost>

    <virtualhost>
        <name>test</name>
        <test>
            <store>
                <!--<class>org.apache.qpid.server.store.MemoryMessageStore</class>
                <class>org.apache.qpid.server.store.derby.DerbyMessageStore</class
                <environment-path>${QPID_WORK}/derbystore/test</environment-path>
            </store>

            <queues>
                <minimumAlertRepeatGap>30000</minimumAlertRepeatGap>
                <maximumMessageCount>50</maximumMessageCount>
                <queue>
                    <name>queue</name>
                    <queue>
                        <exchange>amq.direct</exchange>
                        <maximumQueueDepth>4235264</maximumQueueDepth>
                        <!-- 4Mb -->
                        <maximumMessageSize>2117632</maximumMessageSize>
                        <!-- 2Mb -->
                        <maximumMessageAge>600000</maximumMessageAge>
                        <!-- 10 mins -->
                    </queue>
                </queue>
                <queue>
                    <name>ping</name>
                    <ping>
                        <exchange>amq.direct</exchange>
                        <maximumQueueDepth>4235264</maximumQueueDepth>
                        <!-- 4Mb -->
                        <maximumMessageSize>2117632</maximumMessageSize>
                        <!-- 2Mb -->
                        <maximumMessageAge>600000</maximumMessageAge>
                        <!-- 10 mins -->
                    </ping>
                </queue>
            </queues>
        </test>
    </virtualhost>
</virtualhosts>
```

Chapter 15. Miscellaneous

15.1. JVM Installation verification

15.1.1. Verify JVM on Windows

Firstly confirm that the JAVA_HOME environment variable is set correctly by typing the following at the command prompt:

```
echo %JAVA_HOME%
```

If JAVA_HOME is set you will see something similar to the following:

```
c:"\PROGRA~1"\Java\jdk1.6.0_24\
```

Then confirm that a Java installation (1.6 or higher) is available:

```
java -version
```

If java is available on the path, output similar to the following will be seen:

```
java version "1.6.0_24"
Java(TM) SE Runtime Environment (build 1.6.0_24-b07)
Java HotSpot(TM) 64-Bit Server VM (build 19.1-b02, mixed mode)
```

15.1.2. Verify JVM on Unix

Firstly confirm that the JAVA_HOME environment variable is set correctly by typing the following at the command prompt:

```
echo $JAVA_HOME
```

If JAVA_HOME is set you will see something similar to the following:

```
/usr/java/jdk1.6.0_35
```

Then confirm that a Java installation (1.6 or higher) is available:

```
java -version
```

If java is available on the path, output similar to the following will be seen:

```
java version "1.6.0_35"
Java(TM) SE Runtime Environment (build 1.6.0_35-b10-428-11M3811)
Java HotSpot(TM) 64-Bit Server VM (build 20.10-b01-428, mixed mode)
```

Appendix A. Environment Variables

The following table describes the environment variables understood by the Qpid scripts contained within the `/bin` directory within the Broker distribution.

To take effect, these variables must be set within the shell (and exported - if using Unix) before invoking the script.

Table A.1. Environment variables

Environment variable	Default	Purpose
QPID_HOME	None	The variable used to tell the Broker its installation directory. It must be an absolute path. This is used to determine the location of Qpid's dependency JARs and some configuration files. Typically the value of this variable will look similar to `c:\qpid\qpid-broker-0.28` (Windows) or `/usr/local/qpid/qpid-broker-0.28` (Unix). The installation prefix will differ from installation to installation. If not set, a value for `QPID_HOME` is derived from the location of the script itself.
QPID_WORK	User's home directory	Used as the default root directory for any data written by the Broker. This is the default location for any message data written to persistent stores and the Broker's log file. For example, `QPID_WORK=/var/qpidwork`.
QPID_OPTS	None	This is the preferred mechanism for passing Java system properties to the Broker. The value must be a list of system properties each separate by a space. `-D`*`name1`*`=`*`value1`* `-D`*`name2`*`=`*`value2`*.
QPID_JAVA_GC	`-XX:+HeapDumpOnOutOfMemoryError -XX:+UseConcMarkSweepGC`	This is the preferred mechanism for customising garbage collection behaviour. The value should contain valid garbage collection options(s) for the target JVM. Refer to the JVM's documentation for details.

Environment variable	**Default**	**Purpose**
QPID_JAVA_MEM	`-Xmx1024m`	This is the preferred mechanism for customising the size of the JVM's heap memory. The value should contain valid memory option(s) for the target JVM. Oracle JVMs understand `-Xmx` to specify a maximum heap size and `-Xms` an initial size. For example, `QPID_JAVA_MEM=-Xmx6g` would set a maximum heap size of 6GB. Refer to the JVM's documentation for details.
JAVA_OPTS	None	This is the preferred mechanism for passing any other JVM options. This variable is commonly used to pass options for diagnostic purposes, for instance to turn on verbose GC. `-verbose:gc`. Refer to the JVM's documentation for details.

Appendix B. System Properties

The following table describes the Java system properties used by the Broker to control various optional behaviours.

The preferred method of enabling these system properties is using the `QPID_OPTS` environment variable described in the previous section.

Table B.1. System properties

System property	Default	Purpose
qpid.shared_msg_group	`qpid.no-group`	Controls the default group for groups operating in 'shared groups' mode. See Section 9.1.7, “ Messaging Grouping ”
qpid.broker_heartbeat_timeout_factor	2	Factor to determine the maximum length of that may elapse between heartbeats being received from the peer before a connection is deemed to have been broken.
qpid.broker_dead_letter_exchange_suffix	_DLE	Used with the Section 12.5.3, “Dead Letter Queues (DLQ)” feature. Governs the suffix used when generating a name for a Dead Letter Exchange.
qpid.broker_dead_letter_queue_suffix	_DLQ	Used with the Section 12.5.3, “Dead Letter Queues (DLQ)” feature. Governs the suffix used when generating a name for a Dead Letter Queue.
qpid.broker_msg_auth	false	If set true, the Broker ensures that the user id of each received message matches the user id of the producing connection. If this check fails, the message is returned to the producer's connection with a 403 (Access Refused) error code. This is check is currently not enforced when using AMQP 0-10 and 1-0 protocols.
qpid.broker_status_updates	true	If set true, the Broker will produce operational logging messages.
qpid.broker_default_amqp_protocol_excludes	none	Controls the AMQP protocol versions supported by the Broker. The value of this property is a comma separated list of AMQP protocol versions whose support should be disabled.
qpid.broker_default_amqp_protocol_includes	none	Controls the AMQP protocol versions supported by the Broker. The value of this property is a comma separated

System property	**Default**	**Purpose**
		list of AMQP protocol versions whose support should be enabled.
qpid.broker_default_supported_protocol_version_reply	none	Used during protocol negotiation. If set, the Broker will offer this AMQP version to a client requesting an AMQP protocol that is not supported by the Broker. If not set, the Broker offers the highest protocol version it supports.
qpid.broker_disabled_features	none	Allows optional Broker features to be disabled. Currently understood feature names are: `qpid.jms-selector` Feature names should be comma separated.
qpid.broker_frame_size	65536	Maximum AMQP frame size supported by the Broker.
qpid.broker_jmx_method_rights_infer_all_access	true	Used when using ACLs and the JMX management interface. If set true, the METHOD object permission is sufficient to allow the user to perform the operation. If set false, the user will require both the METHOD object permission and the underlying object permission too (for instance QUEUE object permission if performing management operations on a queue). If the user is not granted both permissions, the operation will be denied.
qpid.broker_jmx_use_custom_rmi_socket_factory	true	Applicable to the JMX management interface. If true, the Broker creates a custom RMI socket factory that is secured from changes outside of the JVM. If false, a standard RMI socket factory is used. It is not recommended to change this property from its default value.
qpid.broker_log_records_buffer_size	4096	Controls the number of recent Broker log entries that remain viewable online via the HTTP Management interface.

Appendix C. Operational Logging

The Broker will, by default, produce structured log messages in response to key events in the lives of objects within the Broker. These consise messages are designed to allow the user to understand the actions of the Broker in retrospect. This is valuable for problem diagnosis and provides a useful audit trail.

Each log message includes details of the entity causing the action (e.g. a management user or messaging client connection), the entity receiving the action (e.g. a queue or connection) and a description of operation itself.

The log messages have the following format:

```
    [Actor] {[Subject]} [Message Id] [Message Text]
```

Where:

- `Actor` is the entity within the Broker that is *performing* the action. There are actors corresponding to the Broker itself, Management, Connection, and Channels. Their format is described in the table below.
- `Subject` (optional) is the entity within the Broker that is *receiving* the action. There are subjects corresponding to the Connections, Channels, Queues, Exchanges, Subscriptions, and Message Stores. Their format is described in the table below.

  Some actions are reflexive, in these cases the Actor and Subject will be equal.
- `Message Id` is an identifier for the type of message. It has the form three alphas and four digits separated by a hyphen `AAA-9999`.
- `Message Text` is a textual description

To illustrate, let's look at two examples.

`CON-1001` is used when a messages client makes an AMQP connection. The connection actor (`con`) provides us with details of the peer's connection: the user id used by the client (myapp1), their IP, ephemeral port number and the name of the virtual host. The message text itself gives us further details about the connection: the client id, the protocol version in used, and details of the client's qpid library.

```
[con:8(myapp1@/127.0.0.1:52851/default)] CON-1001 : Open : Client ID : clientid :
              Protocol Version : 0-10 : Client Version : 0.28 : Client Product : qp
```

`QUE-1001` is used when a queue is created. The connection actor `con` tells us details of the connection performing the queue creation: the user id used by the client (myapp1), the IP, ephemeral port number and the name of the virtual host. The queue subject tells use the queue's name (myqueue) and the virtualhost. The message itself tells us more information about the queue that is being created.

```
[con:8(myapp1@/127.0.0.1:52851/default)/ch:0] [vh(/default)/qu(myqueue)] QUE-1001
```

The first two tables that follow describe the actor and subject entities, then the later provide a complete catalogue of all supported messages.

Table C.1. Actors Entities

Actor Type	Format and Purpose
Broker	[Broker]

Actor Type	**Format and Purpose**
	Used during startup and shutdown
Management	[mng:*userid*(*clientip*:*ephemeralport*)]
	Used for operations performed by the either the JMX or Web Management interfaces.
Connection	[con:*connectionnumber*(*userid*@/*clientip*:*ephemeralport*/*virtualhostname*)]
	Used for operations performed by a client connection. Note that connections are numbered by a sequence number that begins at 1.
Channel	[con:*connectionnumber*(*userid*@/*clientip*:*ephemeralport*/*virtualhostname*/ch:*channelnumber*)]
	Used for operations performed by a client's channel (corresponds to the JMS concept of Session). Note that channels are numbered by a sequence number that is scoped by the owning connection.

Table C.2. Subject Entities

Subject Type	**Format and Purpose**
Connection	[con:*connectionnumber*(*userid*@/*clientip*:*ephemeralport*/*virtualhostname*)]
	A connection to the Broker.
Channel	[con:*connectionnumber*(*userid*@/*clientip*:*ephemeralport*/*virtualhostname*/ch:*channelnumber*)]
	A client's channel within a connection.
Subscription	[sub:*subscriptionnumber*(vh(/*virtualhostname*)/qu(*queuename*)]
	A subscription to a queue. This corresponds to the JMS concept of a Consumer.
Queue	[vh(/*virtualhostname*)/qu(*queuename*)]
	A queue on a virtualhost
Exchange	[vh(/*virtualhostname*)/ex(*exchangetype*/*exchangename*)]
	An exchange on a virtualhost
Binding	[vh(/*virtualhostname*)/ex(*exchangetype*/*exchangename*)/qu(*queuename*)/rk(*bindingkey*)]
	A binding between a queue and exchange with the giving binding key.
Message Store	[vh(/*virtualhostname*)/ms(*messagestorename*)]
	A virtualhost/message store on the Broker.

The following tables lists all the operation log messages that can be produced by the Broker, and the describes the circumstances under which each may be seen.

Table C.3. Broker Log Messages

Message Id	Message Text / Purpose
BRK-1001	Startup : Version: *`version`* Build: *`build`*
	Indicates that the Broker is starting up
BRK-1002	Starting : Listening on *`transporttype`* port *`portnumber`*
	Indicates that the Broker has begun listening on a port.
BRK-1003	Shutting down : *`transporttype`* port *`portnumber`*
	Indicates that the Broker has stopped listening on a port.
BRK-1004	Qpid Broker Ready
	Indicates that the Broker is ready for normal operations.
BRK-1005	Stopped
	Indicates that the Broker is stopped.
BRK-1006	Using configuration : *`file`*
	Indicates the name of the configuration store in use by the Broker.
BRK-1007	Using logging configuration : *`file`*
	Indicates the name of the log configuration file in use by the Broker.
BRK-1008	*`delivered|received`* : *`size`* kB/s peak : *`size`* bytes total
	Statistic - bytes delivered or received by the Broker.
BRK-1009	*`delivered|received`* : *`size`* msg/s peak : *`size`* msgs total
	Statistic - messages delivered or received by the Broker.

Table C.4. Management Log Messages

Message Id	Message Text / Purpose
MNG-1001	*`type`* Management Startup
	Indicates that a Management plugin is starting up. Currently supported management plugins are JMX and Web.
MNG-1002	Starting : *`type`* : Listening on port *`port`*
	Indicates that a Management plugin is listening on the given port.
MNG-1003	Shutting down : *`type`* : port *`port`*
	Indicates that a Management plugin is ceasing to listen on the given port.
MNG-1004	*`type`* Management Ready

Message Id	Message Text / Purpose
	Indicates that a Management plugin is ready for work.
MNG-1005	*type* Management Stopped
	Indicates that a Management plugin is stopped.
MNG-1006	Using SSL Keystore : *file*
	Indicates that a Management transport is secured by SSL and using the given keystore file.
MNG-1007	Open : User *username*
	Indicates the opening of a connection to Management has by the given username.
MNG-1008	Close : User *username*
	Indicates the closing of a connection to Management has by the given username.

Table C.5. Virtual Host Log Messages

Message Id	Message Text / Purpose
VHT-1001	Created : *virtualhostname*
	Indicates that a virtualhost has been created.
VHT-1002	Closed
	Indicates that a virtualhost has been closed. This occurs on Broker shutdown.
VHT-1003	*virtualhostname* : *delivered|received* : *size* kB/s peak : *size* bytes total
	Statistic - bytes delivered or received by the virtualhost.
VHT-1004	*virtualhostname* : *delivered|received* : *size* msg/s peak : *size* msgs total
	Statistic - messages delivered or received by the virtualhost.
VHT-1005	Unexpected fatal error
	Virtualhost has suffered an unexpected fatal error, check the logs for more details.

Table C.6. Queue Log Messages

Message Id	Message Text / Purpose
QUE-1001	Create : Owner: *owner* *AutoDelete* [*Durable*] *Transient* Priority: *numberofpriorities*
	Indicates that a queue has been created.
QUE-1002	Deleted
	Indicates that a queue has been deleted.
QUE-1003	Overfull : Size : *size* bytes, Capacity : *maximumsize*
	Indicates that a queue has exceeded its permitted capacity. See Section 12.3.1, “Producer Flow Control” for details.

Message Id	Message Text / Purpose
QUE-1004	Underfull : Size : *size* bytes, Resume Capacity : *resumesize*
	Indicates that a queue has fallen to its resume capacity. See Section 12.3.1, “Producer Flow Control” for details.

Table C.7. Exchange Log Messages

Message Id	Message Text / Purpose
EXH-1001	Create : [*Durable*] Type: *type* Name: *exchange name*
	Indicates that an exchange has been created.
EXH-1002	Deleted
	Indicates that an exchange has been deleted.
EXH-1003	Discarded Message : Name: *exchange name* Routing Key: *routing key*
	Indicates that an exchange received a message that could not be routed to at least one queue. queue has exceeded its permitted capacity. See Section 4.3.4, “Unrouteable Messages” for details.

Table C.8. Binding Log Messages

Message Id	Message Text / Purpose
BND-1001	Create : Arguments : *arguments*
	Indicates that a binding has been made between an exchange and a queue.
BND-1002	Deleted
	Indicates that a binding has been deleted

Table C.9. Connection Log Messages

Message Id	Message Text / Purpose
CHN-1001	Open : Client ID : *clientid* : Protocol Version : *protocol version* : Client Version : *client version* : Client Product :*client product*
	Indicates that a connection has been opened. The Broker logs one of these message each time it learns more about the client as the connection is negotiated.
CHN-1002	Close
	Indicates that a connection has been closed. This message is logged regardless of if the connection is closed normally, or if the connection is somehow lost e.g network error.
CHN-1003	Closed due to inactivity

Message Id	Message Text / Purpose
	Used when heart beating is in-use. Indicates that the connection has not received a heartbeat for too long and is therefore closed as being inactive.

Table C.10. Channel Log Messages

Message Id	Message Text / Purpose
CHN-1001	Create
	Indicates that a channel (corresponds to the JMS concept of Session) has been created.
CHN-1002	Flow Started
	Indicates message flow to a session has begun.
CHN-1003	Close
	Indicates that a channel has been closed.
CHN-1004	Prefetch Size (bytes) `size` : Count `number of messages`
	Indicates the prefetch size in use by a channel.
CHN-1005	Flow Control Enforced (Queue `queue name`)
	Indicates that producer flow control has been imposed on a channel owning to excessive queue depth in the indicated queue. Produces using the channel will be requested to pause the sending of messages. See Section 12.3.1, “Producer Flow Control” for more details.
CHN-1006	Flow Control Removed
	Indicates that producer flow control has been removed from a channel. See Section 12.3.1, “Producer Flow Control” for more details.
CHN-1007	Open Transaction : `time` ms
	Indicates that a producer transaction has been open for longer than that permitted. See Section 12.4, “Producer Transaction Timeout” for more details.
CHN-1008	Idle Transaction : `time` ms
	Indicates that a producer transaction has been idle for longer than that permitted. See Section 12.4, “Producer Transaction Timeout” for more details.
CHN-1009	Discarded message : `message number` as no alternate exchange configured for queue : `queue name`{1} routing key : `routing key`
	Indicates that a channel has discarded a message as the maximum delivery count has been exceeded but the queue defines no alternate exchange. See Section 12.5.2, “Maximum Delivery Count” for more details. Note that `message number` is an internal message reference.
CHN-1010	Discarded message : `message number` as no binding on alternate exchange : `exchange name`

Message Id	Message Text / Purpose
	Indicates that a channel has discarded a message as the maximum delivery count has been exceeded but the queue's alternate exchange has no binding to a queue. See Section 12.5.2, “Maximum Delivery Count” for more details. Note that *`message number`* is an internal message reference.
CHN-1011	Message : *`message number`* moved to dead letter queue : *`queue name`*
	Indicates that a channel has moved a message to the named dead letter queue

Table C.11. Subscription Log Messages

Message Id	Message Text / Purpose
SUB-1001	Create : [*`Durable`*] Arguments : *`arguments`*
	Indicates that a subscription (corresponds to JMS concept of a MessageConsumer) has been created.
SUB-1002	Close
	Indicates that a subscription has been closed.
SUB-1003	State : *`boolean`*
	Indicates that a subscription has changed state. This occurs when the consumer is ready to receive more messages. As this happens frequently in normal operation, this log messages is disabled by default.

Table C.12. Message Store Log Messages

Message Id	Message Text / Purpose
MST-1001	Created
	Indicates that a message store has been created.
MST-1002	Store location : *`path`*
	Indicates that the message store is using *`path`* for the location of the message store.
MST-1003	Closed
	Indicates that the message store has been closed.
MST-1004	Recovery Start
	Indicates that message recovery has begun.
MST-1005	Recovered *`number of messages`* messages.
	Indicates that recovery recovered the given number of messages from the store.
MST-1006	Recovered Complete
	Indicates that the message recovery is concluded.
MST-1007	Store Passivated

Message Id	Message Text / Purpose
	The store is entering a passive state where is it unavailable for normal operations. Currently this message is used by HA when the node is in replica state.
MST-1008	Store overfull, flow control will be enforced
	The store has breached is maximum configured size. See Section 12.3.1, “Producer Flow Control” for details.
MST-1009	Store overfull condition cleared
	The store size has fallen beneath its resume capacity and therefore flow control has been rescinded. See Section 12.3.1, “Producer Flow Control” for details.