AMQP Messaging Broker (Implemented in Java)

AMQP Messaging Broker (Implemented in Java)

Table of Contents

List of Figures

List of Tables

List of Examples

Introduction

Qpid provides two AMQP messaging brokers:

- Implemented in C++ - high performance, low latency, and RDMA support.
- Implemented in Java - Fully JMS compliant, runs on any Java platform.

Both AMQP messaging brokers support clients in multiple languages, as long as the messaging client and the messaging broker use the same version of AMQP.

This manual contains information specific to the broker that is implemented in Java.

Chapter 1. General User Guides

1.1. Java Broker Feature Guide

1.1.1. The Qpid pure Java broker currently supports the following features:

- All features required by the Sun JMS 1.1 specification, fully tested
- Transaction support
- Persistence using a pluggable layer
- Pluggable security using SASL
- Management using JMX and an Eclipse Management Console application
- High performance header-based routing for messages
- Message Priorities
- Configurable logging and log archiving
- Threshold alerting
- ACLs
- Extensively tested on each release, including performance & reliability testing
- Automatic client failover using configurable connection properties
- Durable Queues/Subscriptions

1.1.1.1. Upcoming features:

- Flow To Disk
- IP Whitelist
- AMQP 0-10 Support (for interoperability)

1.2. Qpid Java FAQ

1.2.1. Purpose

Here are a list of commonly asked questions and answers. Click on the the bolded questions for the answer to unfold. If you have any questions which are not on this list, please email our qpid-user list.

1.2.1.1. What is Qpid ?

The java implementation of Qpid is a pure Java message broker that implements the AMQP protocol. Essentially, Qpid is a robust, performant middleware component that can handle your messaging traffic.

It currently supports the following features:

- High performance header-based routing for messages
- All features required by the JMS 1.1 specification. Qpid passes all tests in the Sun JMS compliance test suite
- Transaction support
- Persistence using the high performance Berkeley DB Java Edition. The persistence layer is also pluggable should an alternative implementation be required. The BDB store is available from the ??? page
- Pluggable security using SASL. Any Java SASL provider can be used
- Management using JMX and a custom management console built using Eclipse RCP
- Naturally, interoperability with other clients including the Qpid .NET, Python, Ruby and C++ implementations

1.2.1.2. Why am I getting a ConfigurationException at broker startup ?

1.2.1.2.1. InvocationTargetException

If you get a java.lang.reflect.InvocationTargetException on startup, wrapped as ConfigurationException like this:

```
Error configuring message broker: org.apache.commons.configuration.ConfigurationEx
2008-09-26 15:14:56,529 ERROR [main] server.Main (Main.java:206) - Error configuri
org.apache.commons.configuration.ConfigurationException: java.lang.reflect.Invocat
at org.apache.qpid.server.security.auth.database.ConfigurationFilePrincipalDatabas
at org.apache.qpid.server.security.auth.database.ConfigurationFilePrincipalDatabas
at org.apache.qpid.server.security.auth.database.ConfigurationFilePrincipalDatabas
at org.apache.qpid.server.registry.ConfigurationFileApplicationRegistry.initialise
at org.apache.qpid.server.registry.ApplicationRegistry.initialise(ApplicationRegis
at org.apache.qpid.server.registry.ApplicationRegistry.initialise(ApplicationRegis
at org.apache.qpid.server.Main.startup(Main.java:260)
at org.apache.qpid.server.Main.execute(Main.java:196)
at org.apache.qpid.server.Main.<init>(Main.java:96)
at org.apache.qpid.server.Main.main(Main.java:454)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.ja
at java.lang.reflect.Method.invoke(Method.java:597)
at com.intellij.rt.execution.application.AppMain.main(AppMain.java:90)
Caused by: java.lang.reflect.InvocationTargetException
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.ja
at java.lang.reflect.Method.invoke(Method.java:597)
at org.apache.qpid.server.security.auth.database.ConfigurationFilePrincipalDatabas
```

.. then it means you have a missing password file.

You need to create a password file for your deployment and update your config.xml to reflect the location of the password file for your instance.

The config.xml can be a little confusing in terms of element names and file names for passwords.

To do this, you need to edit the passwordDir element for the broker, which may have a comment to that effect:

```
<passwordDir><!-- Change to the location --></passwordDir>
```

The file should be named passwd by default but if you want to you can change this by editing this element:

```
<value>${passwordDir}/passwd</value>
```

1.2.1.2.2. Cannot locate configuration source null/virtualhosts.xml

If you get this message, wrapped inside a ConfigurationException then you've come across a known issue, see JIRA ???

The work around is to use a qualified path as the parameter value for your -c option, rather than (as you migth be) starting the broker from your installed etc directory. Even going up one level and using a path relative to your Â£QPID_HOME directory would sort this e.g qpid-server -c ./etc/myconfig.xml

1.2.1.3. How do I run the Qpid broker ?

The broker comes with a script for unix/linux/cygwin called qpid-server, which can be found in the bin directory of the installed package. This command can be executed without any paramters and will then use the default configuration file provided on install.

For the Windows OS, please use qpid-server.bat.

There's no need to set your classpath for QPID as the scripts take care of that by adding jar's with classpath defining manifest files to your classpath.

For more information on running the broker please see our ??? page.

1.2.1.4. How can I create a connection using a URL ?

Please see the ??? documentation.

1.2.1.5. How do I represent a JMS Destination string with QPID ?

1.2.1.5.1. Queues

A queue can be created in QPID using the following URL format.

direct://amq.direct/<Destination>/<Queue Name>

For example: direct://amq.direct/<Destination>/simpleQueue

Queue names may consist of any mixture of digits, letters, and underscores.

The ??? is described in more detail on it's own page.

1.2.1.5.2. Topics

A topic can be created in QPID using the following URL format.

topic://amq.topic/<Topic Subscription>/

The topic subscription may only contain the letters A-Z and a-z and digits 0-9.

The topic subscription is formed from a series of words that may only contain the letters A-Z and a-z and digits 0-9. The words are delimited by dots. Each dot represents a new level.

For example: stocks.nyse.ibm

Wildcards can be used on subscription with the following meaning.

- match a single level # match zero or more levels

For example: With two clients 1 - stocks.*.ibm 2 - stocks.#.ibm

Publishing stocks.nyse.ibm will be received by both clients but stocks.ibm and stocks.world.us.ibm will only be received by client 2.

The topic currently does not support wild cards.

1.2.1.6. How do I connect to the broker using JNDI ?

see ???

1.2.1.7. I'm using Spring and Weblogic - can you help me with the configuration for moving over to Qpid ?

Here is a donated Spring configuration file appContext.zip [http://qpid.apache.org/qpid-java-faq.data/appContext.zip] which shows the config for Qpid side by side with Weblogic. HtH !

1.2.1.8. How do I configure the logging level for Qpid ?

The system property

```
amqj.logging.level
```

can be used to configure the logging level. For the broker, you can use the environment variable AMQJ_LOGGING_LEVEL which is picked up by the qpid-run script (called by qpid-server to start the broker) at runtime.

For client code that you've written, simply pass in a system property to your command line to set it to the level you'd like i.e.

```
-Damqj.logging.level=INFO
```

The log level for the broker defaults to INFO if the env variable is not set, but you may find that your log4j properties affect this. Setting the property noted above should address this.

1.2.1.9. How can I configure my application to use Qpid client logging?

If you don't already have a logging implementation in your classpath you should add slf4-log4j12-1.4.0.jar and log4j-1.2.12.jar.

1.2.1.10. How can I configure the broker ?

The broker configuration is contained in the <installed-dir>/etc/config.xml file. You can copy and edit this file and then specify your own configuration file as a parameter to the startup script using the -c flag i.e. qpid-server -c <your_config_file's_path>

For more detailed information on configuration, please see ???

1.2.1.11. What ports does the broker use?

The broker defaults to use port 5672 at startup for AMQP traffic. If the management interface is enabled it starts on port 8999 by default.

The JMX management interface actually requires 2 ports to operate, the second of which is indicated to the client application during connection initiation to the main (default: 8999) port. Previously this second port has been chosen at random during broker startup, however since Qpid 0.5 this has been fixed to a port 100 higher than the main port(ie Default:9099) in order to ease firewall navigation.

1.2.1.12. How can I change the port the broker uses at runtime ?

The broker defaults to use port 5672 at startup for AMQP traffic. The broker also uses port 8999 for the JMX Management interface.

To change the AMQP traffic port use the -p flag at startup. To change the management port use -m i.e. qpid-server -p <port_number_to_use> -m <port_number_to_use>

Use this to get round any issues on your host server with port 5672/8999 being in use/unavailable.

For additional details on what ports the broker uses see Section 1.2.1.11, " What ports does the broker use? " FAQ entry. For more detailed information on configuration, please see ???

1.2.1.13. What command line options can I pass into the qpid-server script ?

The following command line options are available:

The following options are available:

Table 1.1. Command Line Options

Option	Long Option	Description
b	bind	Bind to the specified address overriding any value in the config file
c	config	Use the given configuration file
h	help	Prints list of options
l	logconfig	Use the specified log4j.xml file rather than that in the etc directory
m	mport	Specify port to listen on for the JMX Management. Overrides value in config file
p	port	Specify port to listen on. Overrides value in config file
v	version	Print version information and exit
w	logwatch	Specify interval for checking for logging config changes. Zero means no checking

1.2.1.14. How do I authenticate with the broker ? What user id & password should I use ?

You should login as user guest with password guest

1.2.1.15. How do I create queues that will always be instantiated at broker startup ?

You can configure queues which will be created at broker startup by tailoring a copy of the virtualhosts.xml file provided in the installed qpid-version/etc directory.

So, if you're using a queue called 'devqueue' you can ensure that it is created at startup by using an entry something like this:

```
<virtualhosts>
  <default>test</default>
  <virtualhost>
   <name>test</name>
   <test>
   <queue>
      <name>devqueue</name>
      <devqueue>
        <exchange>amq.direct</exchange>
        <maximumQueueDepth>4235264</maximumQueueDepth>  <!-- 4Mb -->
        <maximumMessageSize>2117632</maximumMessageSize> <!-- 2Mb -->
        <maximumMessageAge>600000</maximumMessageAge>  <!-- 10 mins -->
      </devqueue>
   </queue>
   </test>
  </virtualhost>
</virtualhosts>
```

Note that the name (in thie example above the name is 'test') element should match the virtualhost that you're using to create connections to the broker. This is effectively a namespace used to prevent queue name clashes etc. You can also see that we've set the 'test' virtual host to be the default for any connections which do not specify a virtual host (in the <default> tag).

You can amend the config.xml to point at a different virtualhosts.xml file by editing the <virtualhosts/> element.

So, for example, you could tell the broker to use a file in your home directory by creating a new config.xml file with the following entry:

<virtualhosts>/home/myhomedir/virtualhosts.xml</virtualhosts>

You can then pass this amended config.xml into the broker at startup using the -c flag i.e. qpid-server -c <path>/config.xml

1.2.1.16. How do I create queues at runtime?

Queues can be dynamically created at runtime by creating a consumer for them. After they have been created and bound (which happens automatically when a JMS Consumer is created) a publisher can send messages to them.

1.2.1.17. How do I tune the broker?

There are a number of tuning options available, please see the Section 2.10, “ How to Tune M3 Java Broker Performance ” page for more information.

1.2.1.18. Where do undeliverable messages end up ?

At present, messages with an invalid routing key will be returned to the sender. If you register an exception listener for your publisher (easiest to do by making your publisher implement the ExceptionListener interface and coding the onException method) you'll see that you end up in onException in this case. You can expect to be catching a subclass of org.apache.qpid.AMQUndeliveredException.

1.2.1.19. Can I configure the name of the Qpid broker log file at runtime ?

If you simply start the Qpid broker using the default configuration, then the log file is written to $QPID_WORK/log/qpid.log

This is not ideal if you want to run several instances from one install, or acrhive logs to a shared drive from several hosts.

To make life easier, there are two optional ways to configure the naming convention used for the broker log.

1.2.1.19.1. Setting a prefix or suffix

Users should set the following environment variables before running qpid-server:

QPID_LOG_PREFIX - will prefix the log file name with the specified value e.g. if you set this value to be the name of your host (for example) it could look something like host123qpid.log

QPID_LOG_SUFFIX - will suffix the file name with the specified value e.g. if you set this value to be the name of your application (for example) if could look something like qpidMyApp.log

1.2.1.19.2. Including the PID

Setting either of these variables to the special value PID will introduce the process id of the java process into the file name as a prefix or suffix as specified**

1.2.1.20. My client application appears to have hung?

The client code currently has various timeouts scattered throughout the code. These can cause your client to appear like it has hung when it is actually waiting for the timeout ot compelete. One example is when the broker becomes non-responsive, the client code has a hard coded 2 minute timeout that it will wait when closing a connection. These timeouts need to be consolidated and exposed. see ???

1.2.1.21. How do I contact the Qpid team ?

For general questions, please subscribe to the users@qpid.apache.org [mailto:users@qpid.apache.org] mailing list.

For development questions, please subscribe to the dev@qpid.apache.org [mailto:dev@qpid.apache.org] mailing list.

More details on these lists are available on our ??? page.

1.2.1.22. How can I change a user's password while the broker is up ?

You can do this via the ???. To do this simply log in to the management console as an admin user (you need to have created an admin account in the jmxremote.access file first) and then select the 'UserManagement' mbean. Select the user in the table and click the Set Password button. Alternatively, update the password file and use the management console to reload the file with the button at the bottom of the 'UserManagement' view. In both cases, this will take effect when the user next logs in i.e. will not cause them to be disconnected if they are already connected.

For more information on the Management Console please see our ???

1.2.1.23. How do I know if there is a consumer for a message I am going to send?

Knowing that there is a consumer for a message is quite tricky. That said using the qpid.jms.Session#createProducer with immediate and mandatory set to true will get you part of the way there.

If you are publishing to a well known queue then immediate will let you know if there is any consumer able to pre-fetch that message at the time you send it. If not it will be returned to you on your connection listener.

If you are sending to a queue that the consumer creates then the mandatory flag will let you know if they have not yet created that queue.

These flags will not be able to tell you if the consuming application has received the message and is able to process it.

1.2.1.24. How can I inspect the contents of my MessageStore?

The management console can be used to interogate an active broker and browse the contents of a queue.See the ??? page for further details.

1.2.1.25. Why are my transient messages being so slow?

You should check that you aren't sending persistent messages, this is the default. If you want to send transient messages you must explicitly set this option when instantiating your MessageProducer or on the send() method.

1.2.1.26. Why does my producer fill up the broker with messages?

Switch on producer flow control to prevent temporary spikes in message production over-filling the broker. Of course, if the long-term rate of message production exceeds the rate of message consumption then that is an architectural problem that can only be temporarily mitigated by producer flow control.

1.2.1.27. The broker keeps throwing an OutOfMemory exception?

The broker can no longer store any more messages in memory. This is particular evident if you are using the MemoryMessageStore. To alleviate this issue you should ensure that your clients are consuming all the messages from the broker.

You may also want to increase the memory allowance to the broker though this will only delay the exception if you are publishing messages faster than you are consuming. See ??? for details of changing the memory settings.

1.2.1.28. Why am I getting a broker side exception when I try to publish to a queue or a topic ?

If you get a stack trace like this when you try to publish, then you may have typo'd the exchange type in your queue or topic declaration. Open your virtualhosts.xml and check that the

```
<exchange>amq.direct</exchange>
```

```
2009-01-12 15:26:27,957 ERROR [pool-11-thread-2] protocol.AMQMinaProtocolSession (
java.lang.NullPointerException
        at org.apache.qpid.server.security.access.PrincipalPermissions.authorise(P
        at org.apache.qpid.server.security.access.plugins.SimpleXML.authorise(Simp
        at org.apache.qpid.server.handler.QueueBindHandler.methodReceived(QueueBin
        at org.apache.qpid.server.handler.ServerMethodDispatcherImpl.dispatchQueue
        at org.apache.qpid.framing.amqp_8_0.QueueBindBodyImpl.execute(QueueBindBod
        at org.apache.qpid.server.state.AMQStateManager.methodReceived(AMQStateMan
        at org.apache.qpid.server.protocol.AMQMinaProtocolSession.methodFrameRecei
        at org.apache.qpid.framing.AMQMethodBodyImpl.handle(AMQMethodBodyImpl.java
        at org.apache.qpid.server.protocol.AMQMinaProtocolSession.frameReceived(AM
        at org.apache.qpid.server.protocol.AMQMinaProtocolSession.dataBlockReceive
        at org.apache.qpid.server.protocol.AMQPFastProtocolHandler.messageReceived
        at org.apache.mina.common.support.AbstractIoFilterChain$TailFilter.message
        at org.apache.mina.common.support.AbstractIoFilterChain.callNextMessageRec
        at org.apache.mina.common.support.AbstractIoFilterChain.access$1200(Abstra
        at org.apache.mina.common.support.AbstractIoFilterChain$EntryImpl$1.messag
        at org.apache.qpid.pool.PoolingFilter.messageReceived(PoolingFilter.java:3
        at org.apache.mina.filter.ReferenceCountingIoFilter.messageReceived(Refere
        at org.apache.mina.common.support.AbstractIoFilterChain.callNextMessageRec
        at org.apache.mina.common.support.AbstractIoFilterChain.access$1200(Abstra
        at org.apache.mina.common.support.AbstractIoFilterChain$EntryImpl$1.messag
        at org.apache.mina.filter.codec.support.SimpleProtocolDecoderOutput.flush(
        at org.apache.mina.filter.codec.QpidProtocolCodecFilter.messageReceived(Qp
        at org.apache.mina.common.support.AbstractIoFilterChain.callNextMessageRec
        at org.apache.mina.common.support.AbstractIoFilterChain.access$1200(Abstra
        at org.apache.mina.common.support.AbstractIoFilterChain$EntryImpl$1.messag
        at org.apache.qpid.pool.Event$ReceivedEvent.process(Event.java:86)
        at org.apache.qpid.pool.Job.processAll(Job.java:110)
        at org.apache.qpid.pool.Job.run(Job.java:149)
        at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecut
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.j
        at java.lang.Thread.run(Thread.java:619)
```

1.2.1.29. Why is there a lot of AnonymousIoService threads

These threads are part of the thread pool used by Mina to process the socket. In the future we may provide tuning guidelines but at this point we have seen no performance implications from the current configuration. As the threads are part of a pool they should remain inactive until required.

1.2.1.30. "unable to certify the provided SSL certificate using the current SSL trust store" when connecting the Management Console to the broker.

You have not configured the console's SSL trust store properly, see ??? for more details.

1.2.1.31. Can a use TCP_KEEPALIVE or AMQP heartbeating to keep my connection open?

See ???

1.3. Java Environment Variables

1.3.1. Setting Qpid Environment Variables

1.3.1.1. Qpid Deployment Path Variables

There are two main Qpid environment variables which are required to be set for Qpid deployments, QPID_HOME and QPID_WORK.

QPID_HOME - This variable is used to tell the Qpid broker where it's installed home is, which is in turn used to find dependency JARs which Qpid uses.

QPID_WORK - This variable is used by Qpid when creating all 'writeable' directories that it uses. This includes the log directory and the storage location for any BDB instances in use by your deployment (if you're using persistence with BDB). If you do not set this variable, then the broker will default (in the qpid-server script) to use the current user's homedir as the root directory for creating the writeable locations that it uses.

1.3.1.2. Setting Max Memory for the broker

If you simply start the Qpid broker, it will default to use a -Xmx setting of 1024M for the broker JVM. However, we would recommend that you make the maximum -Xmx heap size available, if possible, of 3Gb (for 32-bit platforms).

You can control the memory setting for your broker by setting the QPID_JAVA_MEM variable before starting the broker e.g. -Xmx3668m . Enclose your value within quotes if you also specify a -Xms value. The value in use is echo'd by the qpid-server script on startup.

1.4. Qpid Troubleshooting Guide

1.4.1. I'm getting a java.lang.UnsupportedClassVersionError when I try to start the broker. What does this mean ?

The QPID broker requires JDK 1.5 or later. If you're seeing this exception you don't have that version in your path. Set JAVA_HOME to the correct version and ensure the bin directory is on your path.

java.lang.UnsupportedClassVersionError: org/apache/qpid/server/Main (Unsupported major.minor version 49.0) at java.lang.ClassLoader.defineClass(Ljava.lang.String;[BIILjava.security.ProtectionDomain;)Ljava.lang.Class;(Unknown Source) at

```
java.security.SecureClassLoader.defineClass(Ljava.lang.String;
[BIILjava.security.CodeSource;)Ljava.lang.Class;(SecureClassLoader.java:123) at
java.net.URLClassLoader.defineClass(Ljava.lang.String;Lsun.misc.Resource;)Ljava.lang.Class;
(URLClassLoader.java:251) at java.net.URLClassLoader.access
$100(Ljava.net.URLClassLoader;Ljava.lang.String;Lsun.misc.Resource;)Ljava.lang.Class;
(URLClassLoader.java:55) at java.net.URLClassLoader$1.run()Ljava.lang.Object;
(URLClassLoader.java:194) at
jrockit.vm.AccessController.do_privileged_exc(Ljava.security.PrivilegedExceptionAction;Ljava.security.AccessControlCo
(Unknown Source) at
jrockit.vm.AccessController.doPrivileged(Ljava.security.PrivilegedExceptionAction;Ljava.security.AccessControlContext
(Unknown Source) at java.net.URLClassLoader.findClass(Ljava.lang.String;)Ljava.lang.Class;
(URLClassLoader.java:187) at java.lang.ClassLoader.loadClass(Ljava.lang.String;Z)Ljava.lang.Class;
(Unknown Source) at sun.misc.Launcher
$AppClassLoader.loadClass(Ljava.lang.String;Z)Ljava.lang.Class;(Launcher.java:274) at
java.lang.ClassLoader.loadClass(Ljava.lang.String;)Ljava.lang.Class; (Unknown Source) at
java.lang.ClassLoader.loadClassFromNative(II)Ljava.lang.Class; (Unknown Source)
```

1.4.2. I'm having a problem binding to the required host:port at broker startup ?

This error probably indicates that another process is using the port you the broker is trying to listen on. If you haven't amended the default configuration this will be 5672. To check what process is using the port you can use 'netstat -an |grep 5672'.

To change the port your broker uses, either edit the config.xml you are using. You can specify an alternative config.xml from the one provided in /etc by using the -c flag i.e. qpid-server -c <my config file path>.

You can also amend the port more simply using the -p option to qpid-server i.e. qpid-server -p <my port number'

1.4.3. I'm having problems with my classpath. How can I ensure that my classpath is ok ?

When you are running the broker the classpath is taken care of for you, via the manifest entries in the launch jars that the qpid-server configuration file adds to the classpath.

However, if you are running your own client code and experiencing classspath errors you need to ensure that the client-launch.jar from the installed Qpid lib directory is on your classpath. The manifest for this jar includes the common-launch.jar, and thus all the code you need to run a client application.

1.4.4. I can't get the broker to start. How can I diagnose the problem ?

Firstly have a look at the broker log file - either on stdout or in $QPID_WORK/log/qpid.log or in $HOME/log/qpid.log if you haven't set QPID_WORK.

You should see the problem logged in here via log4j and a stack trace. Have a look at the other entries on this page for common problems. If the log file includes a line like:

"2006-10-13 09:58:14,672 INFO [main] server.Main (Main.java:343) - Qpid.AMQP listening on non-SSL address 0.0.0.0/0.0.0.0:5672"

... then you know the broker started up. If not, then it didn't.

1.4.5. When I try to send messages to a queue I'm getting a error as the queue does not exist. What can I do ?

In Qpid queues need a consumer before they really exist, unless you have used the virtualhosts.xml file to specify queues which should always be created at broker startup. If you don't want to use this config, then simply ensure that you consume first from queue before staring to publish to it. See the entry on our ??? for more details of using the virtualhosts.xml route.

1.5. Broker Configuration Guide

1.5.1. Producer Flow Control

1.5.1.1. 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.

1.5.1.2. Server Configuration

1.5.1.2.1. 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".

```
<queue>
    <name>test</name>
    <test>
        <exchange>amq.direct</exchange>
        <capacity>10485760</capacity>                     <!-- set the queue capac
        <flowResumeCapacity>8388608</flowResumeCapacity>  <!-- set the resume capa
    </test>
</queue>
```

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

```
<virtualhosts>
    <virtualhost>
        <name>localhost</name>
        <localhost>
            <capacity>10485760</capacity>                     <!-- set the queue c
            <flowResumeCapacity>8388608</flowResumeCapacity>  <!-- set the resume 
        </localhost>
    </virtualhost>
</virtualhosts>
```

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.

1.5.1.2.1.1. 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.

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

```
<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>
```

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.

1.5.1.2.2.1. 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
```

1.5.1.3. 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
```

1.5.1.3.1. 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.

1.6. High Availability

1.6.1. 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.

1.6.2. 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.

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

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**.

1.6.3. Two Node Cluster

1.6.3.1. 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 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.

1.6.3.2. Depictions of cluster operation

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

Figure 1.1. Key for figures

1.6.3.2.1. 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 1.2. Normal operation of a two-node cluster

1.6.3.2.2. 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.

Figure 1.3. Failure of master and recovery sequence

1.6.3.2.3. 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 1.4. Failure of replica and subsequent recovery sequence

1.6.3.2.4. 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 1.5. Partition of the network separating master and replica

1.6.3.2.5. 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 1.6. Split Brain

1.6.4. Multi Node Cluster

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

1.6.5. Configuring a Virtual Host to be a node

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

```
<virtualhost>
  <name>myhost</name>
  <myvhost>
    <store>
```

```
      <class>org.apache.qpid.server.store.berkeleydb.BDBHAMessageStore</class>
      <environment-path>${work}/bdbhastore</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>
    ...
 </myvhost>
</virtualhost>
```

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.

1.6.5.1. 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 using the `envConfig` element, and replication config using `repConfig`.

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

```
       ...
    </highAvailability>
    <envConfig>
      <name>je.cleaner.threads</name>
      <value>2</value>
    </envConfig>
    <envConfig>
      <name>je.txn.timeout</name>
      <value>15 min</value>
    </envConfig>
    ...
  </store>
```

And to override the BDB replication configuration options `je.rep.electionsPrimaryRetries`.

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

1.6.6. 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.

1.6.6.1. 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].

1.6.6.2. Coalescing-sync

If enabled (the default) Qpid works to reduce the number of separate file-system sync [http://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.

1.6.6.3. 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.

1.6.6.4. Examples

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

Table 1.2. 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 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.

1.6.7. 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] of book Programming In Apache Qpid [../../Programming-In-Apache-Qpid/html/].

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 1.1. 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''
```

1.6.8. 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.

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
HelperHostPort	String	Read only	Host/port used to allow a new node

Name	Type	Accessibility	Description
			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

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 1.7. BDBHAMessageStore view from jconsole.

Example 1.2. 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
```

1.6.9. 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 1.3. Using `DbPing` utility for monitoring HA nodes.

java -jar je-5.0.58.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: 5.0.58
  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

1.6.10. 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 1.6.5.1, "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.

1.6.11. 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.

1.6.12. 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.

1.6.13. 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 1.4. Performing store backup by using `BDBBackup` class directly

java -cp qpid-bdbstore-0.18.jar org.apache.qpid.server.store.berkeleydb.BDBBackup -fromdir path/to/store/folder -todir path/to/backup/foldeAr

In the example above BDBBackup utility is called from qpid-bdbstore-0.18.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 1.5. 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.

1.6.14. 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 1.6. Enabling replication

java -jar je-5.0.58.jar DbEnableReplication -h /path/to/store -groupName MyReplicationGroup -nodeName MyNode1 -nodeHostPort localhost:5001

In the examples above, je jar of version 5.0.58 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 1.7. 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.

1.6.15. 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 1.8. Reseting of replication group with `DbResetRepGroup`

java -cp je-5.0.58.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 5.0.58 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.

1.6.16. 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.

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.

Test/ Broker	**No HA**	**SYNC, SYNC, ALL**	**WRITE_NO_SYNC, WRITE_NO_SYNC, ALL**	**SYNC, SYNC, ALL - coalescing-sync**	**WRITE_NO_SYNC, WRITE_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 1.8. 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 2. How Tos

2.1. Add New Users

The Qpid Java Broker has a single reference source (???) that defines all the users in the system.

To add a new user to the broker the password file must be updated. The details about adding entries and when these updates take effect are dependent on the file format each of which are described below.

2.1.1. Available Password file formats

There are currently two different file formats available for use depending on the PrincipalDatabase that is desired. In all cases the clients need not be aware of the type of PrincipalDatabase in use they only need support the SASL mechanisms they provide.

- Section 2.1.1.1, “ Plain ”
- Section 2.1.1.3, “ Base64MD5 Password File Format ”

2.1.1.1. Plain

The plain file has the following format:

```
# Plain password authentication file.
# default name : passwd
# Format <username>:<password>
#e.g.
martin:password
```

As the contents of the file are plain text and the password is taken to be everything to the right of the ':'(colon). The password, therefore, cannot contain a ':' colon, but this can be used to delimit the password.

Lines starting with a '#' are treated as comments.

2.1.1.2. Where is the password file for my broker ?

The location of the password file in use for your broker is as configured in your config.xml file.

```
<principal-databases>
            <principal-database>
                <name>passwordfile</name>
                <class>org.apache.qpid.server.security.auth.database.PlainPassword
                <attributes>
                    <attribute>
                        <name>passwordFile</name>
                        <value>${conf}/passwd</value>
                    </attribute>
                </attributes>
            </principal-database>
        </principal-databases>
```

So in the example config.xml file this password file lives in the directory specified as the conf directory (at the top of your config.xml file).

If you wish to use Base64 encoding for your password file, then in the <class> element above you should specify org.apache.qpid.server.security.auth.database.Base64MD5PasswordFilePrincipalDatabase

The default is:

```
 <conf>${prefix}/etc</conf>
```

2.1.1.3. Base64MD5 Password File Format

This format can be used to ensure that SAs cannot read the plain text password values from your password file on disk.

The Base64MD5 file uses the following format:

```
# Base64MD5 password authentication file
# default name : qpid.passwd
# Format <username>:<Base64 Encoded MD5 hash of the users password>
#e.g.
martin:X03MO1qnZdYdgyfeuILPmQ==
```

As with the Plain format the line is delimited by a ':'(colon). The password field contains the MD5 Hash of the users password encoded in Base64.

This file is read on broker start-up and is not re-read.

2.1.1.4. How can I update a Base64MD5 password file ?

To update the file there are two options:

1. Edit the file by hand using the *qpid-passwd* tool that will generate the required lines. The output from the tool is the text that needs to be copied in to your active password file. This tool is located in the broker bin directory. Eventually it is planned for this tool to emulate the functionality of ??? for qpid passwd files. *NOTE:* For the changes to be seen by the broker you must either restart the broker or reload the data with the management tools (see ???)

2. Use the management tools to create a new user. The changes will be made by the broker to the password file and the new user will be immediately available to the system (see ???).

2.1.2. Dynamic changes to password files.

The Plain password file and the Base64MD5 format file are both only read once on start up.

To make changes dynamically there are two options, both require administrator access via the Management Console (see ???)

1. You can replace the file and use the console to reload its contents.
2. The management console provides an interface to create, delete and amend the users. These changes are written back to the active password file.

2.1.3. How password files and PrincipalDatabases relate to authentication mechanisms

For each type of password file a PrincipalDatabase exists that parses the contents. These PrincipalDatabases load various SASL mechanism based on their supportability. e.g. the Base64MD5 file format can't support Plain authentication as the plain password is not available. Any client connecting need only be concerned about the SASL module they support and not the type of PrincipalDatabase. So I client that understands CRAM-MD5 will work correctly with a Plain and Base64MD5 PrincipalDatabase.

Table 2.1. File Format and Principal Database

FileFormat/PrincipalDatabase	SASL
Plain	AMQPLAIN PLAIN CRAM-MD5
Base64MD5	CRAM-MD5 CRAM-MD5-HASHED

For details of SASL support see ???

2.2. Configuring ACLs

In Qpid, Access Control Lists (ACLs) specify which actions can be performed by each authenticated user. To enable, the <acl/> element is used within the <security/> element of the configuration XML. In the Java Broker, the ACL may be imposed broker wide or applied to individual virtual hosts. The <acl/> configuration references a text file containing the ACL rules. By convention, this file should have a .acl extension.

2.2.1. Enabling ACLs

To apply an ACL broker-wide, add the following to the config.xml (assuming that *conf* has been set to a suitable location such as ${QPID_HOME}/etc):

```
<broker>
  ...
  <security>
    ...
    <acl>${conf}/broker.acl</acl>
  </security>
</broker>
```

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

```
<virtualhost>
```

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

2.2.2. 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.

2.2.3. 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 2.2. ACL Rules: 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 2.3. ACL Rules:action

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

Table 2.4. ACL Rules:object

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 (not currently used in Java Broker)

Table 2.5. ACL Rules:property

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 2.6. ACL rules:components (Java Broker only)

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	

2.2.4. 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.

2.2.4.1. 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
```

2.2.4.2. 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
```

2.2.4.3. 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
```

2.2.4.4. 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
```

2.3. Configuring 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 to allowing assignment of GroupPrincipals for a given authenticated user.

2.3.1. FileGroupManager

The FileGroupManager allows specifying group membership in a flat file on disk, and is also exposed for inspection and update through the brokers HTTP management interface.

To enable the FileGroupManager, add the following configuration to the config.xml, adjusting the groupFile attribute value to match your desired groups file location.

```
    ...
    <security>
        <file-group-manager>
            <attributes>
              <attribute>
                <name>groupFile</name>
                 <value>${conf}/groups</value>
              </attribute>
            </attributes>
        </file-group-manager>
    </security>
    ...
```

2.3.1.1. 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.

2.4. Configure Java Qpid to use a SSL connection.

2.4.1. Using SSL connection with Qpid Java.

This section will show how to use SSL to enable secure connections between a Java client and broker.

2.4.2. Setup

2.4.2.1. Broker Setup

The broker configuration file (config.xml) needs to be updated to include the SSL keystore location details.

```
<!-- Additions required to Connector Section -->

<ssl>
    <enabled>true</enabled>
    <sslOnly>true</sslOnly>
    <keyStorePath>/path/to/keystore.ks</keyStorePath>
    <keyStorePassword>keystorepass</keyStorePassword>
</ssl>
```

The sslOnly option is included here for completeness however this will disable the unencrypted port and leave only the SSL port listening for connections.

2.4.2.2. Client Setup

The best place to start looking is class *SSLConfiguration* this is provided to the connection during creation however there is currently no example that demonstrates its use.

2.4.3. Performing the connection.

2.5. Configure Log4j CompositeRolling Appender

2.5.1. How to configure the CompositeRolling log4j Appender

There are several sections of our default log4j file that will need your attention if you wish to fully use this Appender.

1. Enable the Appender

   The default log4j.xml file uses the FileAppender, swap this for the ArchivingFileAppender as follows:

```
<!-- Log all info events to file -->
<root>
    <priority value="info"/>

    <appender-ref ref="ArchivingFileAppender"/>
</root>
```

2. Configure the Appender

   The Appender has a number of parameters that can be adjusted depending on what you are trying to achieve. For clarity lets take a quick look at the complete default appender:

```
<appender name="ArchivingFileAppender" class="org.apache.log4j.QpidCompositeR
      <!-- Ensure that logs allways have the dateFormat set-->
      <param name="StaticLogFileName" value="false"/>
      <param name="File" value="${QPID_WORK}/log/${logprefix}qpid${logsuffix}
      <param name="Append" value="false"/>
      <!-- Change the direction so newer files have bigger numbers -->
      <!-- So log.1 is written then log.2 etc This prevents a lot of file ren
      <param name="CountDirection" value="1"/>
      <!-- Use default 10MB -->
      <!--param name="MaxFileSize" value="100000"/-->
      <param name="DatePattern" value="'.'yyyy-MM-dd-HH-mm"/>
      <!-- Unlimited number of backups -->
      <param name="MaxSizeRollBackups" value="-1"/>
      <!-- Compress(gzip) the backup files-->
      <param name="CompressBackupFiles" value="true"/>
      <!-- Compress the backup files using a second thread -->
      <param name="CompressAsync" value="true"/>
      <!-- Start at zero numbered files-->
      <param name="ZeroBased" value="true"/>
      <!-- Backup Location -->
      <param name="backupFilesToPath" value="${QPID_WORK}/backup/log"/>

      <layout class="org.apache.log4j.PatternLayout">
          <param name="ConversionPattern" value="%d %-5p [%t] %C{2} (%F:%L) -
      </layout>
```

```
    </appender>
```

The appender configuration has three groups of parameter configuration.

The first group is for configuration of the file name. The default is to write a log file to QPID_WORK/log/qpid.log (Remembering you can use the logprefix and logsuffix values to modify the file name, see Property Config).

```
        <!-- Ensure that logs always have the dateFormat set-->
        <param name="StaticLogFileName" value="false"/>
        <param name="File" value="${QPID_WORK}/log/${logprefix}qpid${logsuffix}
        <param name="Append" value="false"/>
```

The second section allows the specification of a Maximum File Size and a DatePattern that will be used to move on to the next file.

When MaxFileSize is reached a new log file will be created The DataPattern is used to decide when to create a new log file, so here a new file will be created for every minute and every 10Meg of data. So if 15MB of data is made every minute then there will be two log files created each minute. One at the start of the minute and a second when the file hit 10MB. When the next minute arrives a new file will be made even though it only has 5MB of content. For a production system it would be expected to be changed to something like 'yyyy-MM-dd' which would make a new log file each day and keep the files to a max of 10MB.

The final MaxSizeRollBackups allows you to limit the amount of disk you are using by only keeping the last n backups.

```
        <!-- Change the direction so newer files have bigger numbers -->
        <!-- So log.1 is written then log.2 etc This prevents a lot of file rer
        <param name="CountDirection" value="1"/>
        <!-- Use default 10MB -->
        <!--param name="MaxFileSize" value="100000"/-->
        <param name="DatePattern" value="'.'yyyy-MM-dd-HH-mm"/>
        <!-- Unlimited number of backups -->
        <param name="MaxSizeRollBackups" value="-1"/>
```

The final section allows the old log files to be compressed and copied to a new location.

```
        <!-- Compress(gzip) the backup files-->
        <param name="CompressBackupFiles" value="true"/>
        <!-- Compress the backup files using a second thread -->
        <param name="CompressAsync" value="true"/>
        <!-- Start at zero numbered files-->
        <param name="ZeroBased" value="true"/>
        <!-- Backup Location -->
        <param name="backupFilesToPath" value="${QPID_WORK}/backup/log"/>
```

2.6. Configure the Broker via config.xml

2.6.1. Broker config.xml Overview

The broker config.xml file which is shipped in the etc directory of any Qpid binary distribution details various options and configuration for the Java Qpid broker implementation.

In tandem with the virtualhosts.xml file, the config.xml file allows you to control much of the deployment detail for your Qpid broker in a flexible fashion.

Note that you can pass the config.xml you wish to use for your broker instance to the broker using the -c command line option. In turn, you can specify the paths for the broker password file and virtualhosts.xml files in your config.xml for simplicity.

For more information about command line configuration options please see ???.

2.6.2. Qpid Version

The config format has changed between versions here you can find the configuration details on a per version basis.

??? ???

2.7. Configure the Virtual Hosts via virtualhosts.xml

2.7.1. virtualhosts.xml Overview

This configuration file contains details of all queues and topics, and associated properties, to be created on broker startup. These details are configured on a per virtual host basis.

Note that if you do not add details of a queue or topic you intend to use to this file, you must first create a consumer on a queue/topic before you can publish to it using Qpid.

Thus most application deployments need a virtualhosts.xml file with at least some minimal detail.

2.7.1.1. XML Format with Comments

The virtualhosts.xml which currently ships as part of the Qpid distribution is really targeted at development use, and supports various artifacts commonly used by the Qpid development team.

As a result, it is reasonably complex. In the example XML below, I have tried to simplify one example virtual host setup which is possibly more useful for new users of Qpid or development teams looking to simply make use of the Qpid broker in their deployment.

I have also added some inline comments on each section, which should give some extra information on the purpose of the various elements.

```
<virtualhosts>
```

```
        <!-- Sets the default virtual host for connections which do not specify a vh -
        <default>localhost</default>
        <!-- Define a virtual host and all it's config -->
        <virtualhost>
            <name>localhost</name>
            <localhost>
                <!-- Define the types of additional AMQP exchange available for this v
                <!-- Always get amq.direct (for queues) and amq.topic (for topics) by
                <exchanges>
                    <!-- Example of declaring an additional exchanges type for develop
                    <exchange>
                        <type>direct</type>
                        <name>test.direct</name>
                        <durable>true</durable>
                    </exchange>
                </exchanges>

                <!-- Define the set of queues to be created at broker startup -->
                <queues>
                    <!-- The properties configured here will be applied as defaults to
                    <!-- queues subsequently defined unless explicitly overridden -->
                    <exchange>amq.direct</exchange>
                    <!-- Set threshold values for queue monitor alerting to log -->
                    <maximumQueueDepth>4235264</maximumQueueDepth>  <!-- 4Mb -->
                    <maximumMessageSize>2117632</maximumMessageSize> <!-- 2Mb -->
                    <maximumMessageAge>600000</maximumMessageAge>  <!-- 10 mins -->

                    <!-- Define a queue with all default settings -->
                    <queue>
                        <name>ping</name>
                    </queue>
                    <!-- Example definitions of queues with overriden settings -->
                    <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>
    </virtualhosts>
```

2.7.1.2. Using your own virtualhosts.xml

Note that the config.xml file shipped as an example (or developer default) in the Qpid distribution contains an element which defines the path to the virtualhosts.xml.

When using your own virtualhosts.xml you must edit this path to point at the location of your file.

2.8. Configuring Authentication Mechanisms

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 schemesi, each with its own "authentication manager". Different managers may be used on different ports. Each manager has its own configuration element, the presence of which within the <security> section denotes the use of that authentication mechanism. Where only one such manager is configured, that manager will be used on all ports (including JMX). Where more than one authentication manager is configured the configuration must define which manager is the "default", and (if required) the mapping of non-default authentication managers to other ports.

The following configuration sets up three authentication managers, using a password file as the default (e.g. for the JMX port), Kerberos on port 5672 and Anonymous on 5673.

```
<security>
    <pd-auth-manager>
        <principal-database>
            <class>org.apache.qpid.server.security.auth.database.PlainPassword
            <attributes>
                <attribute>
                    <name>passwordFile</name>
                    <value>${conf}/passwd</value>
                </attribute>
            </attributes>
        </principal-database>
    </pd-auth-manager>
    <kerberos-auth-manager><auth-name>sib</auth-name></kerberos-auth-manager>
    <anonymous-auth-manager></anonymous-auth-manager>
    <default-auth-manager>PrincipalDatabaseAuthenticationManager</default-auth
    <port-mappings>
        <port-mapping>
            <port>5672</port>
            <auth-manager>KerberosAuthenticationManager</auth-manager>
        </port-mapping>
        <port-mapping>
            <port>5673</port>
            <auth-manager>AnonymousAuthenticationManager</auth-manager>
        </port-mapping>
    </port-mappings>
</security>
```

2.8.1. Password File

2.8.2. LDAP

```
<security>
    <simple-ldap-auth-manager>
      <provider-url>ldaps://example.com:636/</provider-url>
```

```
            <search-context>dc=example\,dc=com</search-context>
            <search-filter>(uid={0})</search-filter>
        </simple-ldap-auth-manager>
    </security>
```

The authentication manager first connects to the ldap server anonymously and searches for the ldap entity which is identified by the username provided over SASL. Essentially the authentication manager calls DirContext.search(Name name, String filterExpr, Object[] filterArgs, SearchControls cons) with the values of search-context and search-filter as the first two arguments, and the username as the only element in the array which is the third argument.

If the search returns a name from the LDAP server, the AuthenticationManager then attempts to login to the ldap server with the given name and the password.

If the URL to open for authentication is different to that for the search, then the authentication url can be overridden using <provider-auth-url> in addition to providing a <provider-url>. Note that the URL used for authentication should use ldaps:// since passwords will be being sent over it.

By default com.sun.jndi.ldap.LdapCtxFactory is used to create the context, however this can be overridden by specifying <ldap-context-factory> in the configuration.

2.8.3. Kerberos

Kereberos Authentication is configured using the <kerberos-auth-manager> element within the <security> section. When referencing from the default-auth-manager or port-mapping sections, its name is KerberosAuthenticationManager.

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 Manager configuration, and use this for other ports:

```
    <security>
        <pd-auth-manager>
            <principal-database>
                <class>org.apache.qpid.server.security.auth.database.PlainPassword
                <attributes>
                    <attribute>
                        <name>passwordFile</name>
                        <value>${conf}/passwd</value>
                    </attribute>
                </attributes>
            </principal-database>
        </pd-auth-manager>
        <kerberos-auth-manager><auth-name>sib</auth-name></kerberos-auth-manager>
        <default-auth-manager>PrincipalDatabaseAuthenticationManager</default-auth
        <port-mappings>
            <port-mapping>
                <port>5672</port>
                <auth-manager>KerberosAuthenticationManager</auth-manager>
            </port-mapping>
        </port-mappings>
    </security>
```

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

```
    export QPID_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.

2.8.4. SSL Client Certificates

2.8.5. Anonymous

2.9. Debug using log4j

2.9.1. Debugging with log4j configurations

Unfortunately setting of logging in the Java Broker is not simply a matter of setting one of WARN,INFO,DEBUG. At some point in the future we may have more BAU logging that falls in to that category but more likely is that we will have a varioius config files that can be swapped in (dynamically) to understand what is going on.

This page will be host to a variety of useful configuration setups that will allow a user or developer to extract only the information they are interested in logging. Each section will be targeted at logging in a particular area and will include a full log4j file that can be used. In addition the logging *category* elements will be presented and discussed so that the user can create their own file.

Currently the configuration that is available has not been fully documented and as such there are gaps in what is desired and what is available. Some times this is due to the desire to reduce the overhead in message processing, but sometimes it is simply an oversight. Hopefully in future releases the latter will be addressed but care needs to be taken when adding logging to the 'Message Flow' path as this will have performance implications.

2.9.1.1. Logging Connection State *Deprecated*

deprecation notice Version 0.6 of the Java broker includes ??? functionality which improves upon these messages and as such enabling status logging would be more beneficial. The configuration file has been left here for assistence with broker versions prior to 0.6.

The goals of this configuration are to record:

- New Connections
- New Consumers
- Identify slow consumers
- Closing of Consumers
- Closing of Connections

An additional goal of this configuration is to minimise any impact to the 'message flow' path. So it should not adversely affect production systems.

```
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
    <appender name="FileAppender" class="org.apache.log4j.FileAppender">
        <param name="File" value="${QPID_WORK}/log/${logprefix}qpid${logsuffix}.lo
        <param name="Append" value="false"/>

        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d %-5p [%t] %C{2} (%F:%L) - %m
        </layout>

    </appender>

    <appender name="STDOUT" class="org.apache.log4j.ConsoleAppender">

        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d %-5p [%t] %C{2} (%F:%L) - %m
        </layout>
    </appender>

    <category name="Qpid.Broker">

        <priority value="debug"/>
    </category>


    <!-- Provide warnings to standard output -->
    <category name="org.apache.qpid">
        <priority value="warn"/>
    </category>


    <!-- Connection Logging -->
```

```
        <!-- Log details of client starting connection -->
        <category name="org.apache.qpid.server.handler.ConnectionStartOkMethodHandler"
            <priority value="info"/>
        </category>
        <!-- Log details of client closing connection -->
        <category name="org.apache.qpid.server.handler.ConnectionCloseMethodHandler">
            <priority value="info"/>
        </category>
        <!-- Log details of client responding to be asked to closing connection -->

        <category name="org.apache.qpid.server.handler.ConnectionCloseOkMethodHandler"
            <priority value="info"/>
        </category>


        <!-- Consumer Logging -->
        <!-- Provide details of Consumers connecting-->
        <category name="org.apache.qpid.server.handler.BasicConsumeMethodHandler">
            <priority value="debug"/>
        </category>

        <!-- Provide details of Consumers disconnecting, if the call it-->
        <category name="org.apache.qpid.server.handler.BasicCancelMethodHandler">
            <priority value="debug"/>
        </category>
        <!-- Provide details of when a channel closes to attempt to match to the Consu
        <category name="org.apache.qpid.server.handler.ChannelCloseHandler">
            <priority value="info"/>
        </category>

        <!-- Provide details of Consumers starting to consume-->
        <category name="org.apache.qpid.server.handler.ChannelFlowHandler">
            <priority value="debug"/>
        </category>
        <!-- Provide details of what consumers are going to be consuming-->
        <category name="org.apache.qpid.server.handler.QueueBindHandler">
            <priority value="info"/>
        </category>

        <!-- No way of determining if publish message is returned, client log should s

        <root>
            <priority value="debug"/>
            <appender-ref ref="STDOUT"/>
            <appender-ref ref="FileAppender"/>
        </root>

    </log4j:configuration>
```

2.9.1.2. Debugging My Application

This is the most often asked for set of configuration. The goals of this configuration are to record:

- New Connections
- New Consumers
- Message Publications
- Message Consumption
- Identify slow consumers
- Closing of Consumers
- Closing of Connections

NOTE: This configuration enables message logging on the 'message flow' path so should only be used were message volume is low. *Every message that is sent to the broker will generate at least four logging statements*

```
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
    <appender name="FileAppender" class="org.apache.log4j.FileAppender">
        <param name="File" value="${QPID_WORK}/log/${logprefix}qpid${logsuffix}.lo
        <param name="Append" value="false"/>

        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d %-5p [%t] %C{2} (%F:%L) - %m
        </layout>

    </appender>

    <appender name="STDOUT" class="org.apache.log4j.ConsoleAppender">

        <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern" value="%d %-5p [%t] %C{2} (%F:%L) - %m
        </layout>
    </appender>

    <category name="Qpid.Broker">

        <priority value="debug"/>
    </category>


    <!-- Provide warnings to standard output -->
    <category name="org.apache.qpid">
        <priority value="warn"/>
    </category>


    <!-- Connection Logging -->

    <!-- Log details of client starting connection -->
    <category name="org.apache.qpid.server.handler.ConnectionStartOkMethodHandler"
        <priority value="info"/>
```

```
</category>
<!-- Log details of client closing connection -->
<category name="org.apache.qpid.server.handler.ConnectionCloseMethodHandler">
    <priority value="info"/>
</category>
<!-- Log details of client responding to be asked to closing connection -->

<category name="org.apache.qpid.server.handler.ConnectionCloseOkMethodHandler"
    <priority value="info"/>
</category>

<!-- Consumer Logging -->
<!-- Provide details of Consumers connecting-->
<category name="org.apache.qpid.server.handler.BasicConsumeMethodHandler">
    <priority value="debug"/>
</category>

<!-- Provide details of Consumers disconnecting, if the call it-->
<category name="org.apache.qpid.server.handler.BasicCancelMethodHandler">
    <priority value="debug"/>
</category>
<!-- Provide details of when a channel closes to attempt to match to the Consu
<category name="org.apache.qpid.server.handler.ChannelCloseHandler">
    <priority value="info"/>
</category>

<!-- Provide details of Consumers starting to consume-->
<category name="org.apache.qpid.server.handler.ChannelFlowHandler">
    <priority value="debug"/>
</category>
<!-- Provide details of what consumers are going to be consuming-->
<category name="org.apache.qpid.server.handler.QueueBindHandler">
    <priority value="info"/>
</category>

<!-- No way of determining if publish message is returned, client log should s

<!-- WARNING DO NOT ENABLE THIS IN PRODUCTION -->
<!-- Will generate minimum one log statements per published message -->
<!-- Will generate will log receiving of all body frame, count will vary on si
<!-- Empty Message = no body, Body is up to 64kb of data -->
<!-- Will generate three log statements per recevied message -->

<!-- Log messages flow-->
<category name="org.apache.qpid.server.AMQChannel">

    <priority value="debug"/>
</category>

<root>
    <priority value="debug"/>
    <appender-ref ref="STDOUT"/>
    <appender-ref ref="FileAppender"/>
</root>
```

```
</log4j:configuration>
```

2.10. How to Tune M3 Java Broker Performance

2.10.1. Problem Statement

During destructive testing of the Qpid M3 Java Broker, we tested some tuning techniques and deployment changes to improve the Qpid M3 Java Broker's capacity to maintain high levels of throughput, particularly in the case of a slower consumer than produceer (i.e. a growing backlog).

The focus of this page is to detail the results of tuning & deployment changes trialled.

The successful tuning changes are applicable for any deployment expecting to see bursts of high volume throughput (1000s of persistent messages in large batches). Any user wishing to use these options *must test them thoroughly in their own environment with representative volumes*.

2.10.2. Successful Tuning Options

The key scenario being taregetted by these changes is a broker under heavy load (processing a large batch of persistent messages)can be seen to perform slowly when filling up with an influx of high volume transient messages which are queued behind the persistent backlog. However, the changes suggested will be equally applicable to general heavy load scenarios.

The easiest way to address this is to separate streams of messages. Thus allowing the separate streams of messages to be processed, and preventing a backlog behind a particular slow consumer.

These strategies have been successfully tested to mitigate this problem:

Table 2.7.

Strategy	Result
Seperate connections to one broker for separate streams of messages.	Messages processed successfully, no problems experienced
Seperate brokers for transient and persistent messages.	Messages processed successfully, no problems experienced

Separate Connections Using separate connections effectively means that the two streams of data are not being processed via the same buffer, and thus the broker gets & processes the transient messages while processing the persistent messages. Thus any build up of unprocessed data is minimal and transitory.

Separate Brokers Using separate brokers may mean more work in terms of client connection details being changed, and from an operational perspective. However, it is certainly the most clear cut way of isolating the two streams of messages and the heaps impacted.

2.10.2.1. Additional tuning

It is worth testing if changing the size of the Qpid read/write thread pool improves performance (eg. by setting JAVA_OPTS="-Damqj.read_write_pool_size=32" before running qpid-server). By default this is equal to the number of CPU cores, but a higher number may show better performance with some work loads.

It is also important to note that you should give the Qpid broker plenty of memory - for any serious application at least a -Xmx of 3Gb. If you are deploying on a 64 bit platform, a larger heap is definitely worth testing with. We will be testing tuning options around a larger heap shortly.

2.10.3. Next Steps

These two options have been testing using a Qpid test case, and demonstrated that for a test case with a profile of persistent heavy load following by constant transient high load traffic they provide significant improvment.

However, the deploying project *must* complete their own testing, using the same destructive test cases, representative message paradigms & volumes, in order to verify the proposed mitigation options.

The using programme should then choose the option most applicable for their deployment and perform BAU testing before any implementation into a production or pilot environment.

2.11. Qpid Java Build How To

2.11.1. Build Instructions - General

2.11.1.1. Check out the source

Firstly, check the source for Qpid out of our subversion repository:

???

2.11.1.2. Prerequisites

For the broker code you need JDK 1.5.0_15 or later. You should set JAVA_HOME and include the bin directory in your PATH.

Check it's ok by executing java -v !

If you are wanting to run the python tests against the broker you will of course need a version of python.

2.11.2. Build Instructions - Trunk

Our build system has reverted to ant as of May 2008.

The ant target 'help' will tell you what you need to know about the build system.

2.11.2.1. Ant Build Scripts

Currently the Qpid java project builds using ant.

The ant build system is set up in a modular way, with a top level build script and template for module builds and then a module level build script which inherits from the template.

So, at the top level there are:

Table 2.8.

File	Description
build.xml	Top level build file for the project which defines all the build targets
common.xml	Common properties used throughout the build system
module.xml	Template used by all modules which sets up properties for module builds

Then, in each module subdirectory there is:

Table 2.9.

File	Description
build.xml	Defines all the module values for template properties

2.11.2.2. Build targets

The main build targets you are probably interested in are:

Table 2.10.

Target	Description
build	Builds all source code for Qpid
test	Runs the testsuite for Qpid

So, if you just want to compile everything you should run the build target in the top level build.xml file.

If you want to build an installable version of Qpid, run the archive task from the top level build.xml file.

If you want to compile an individual module, simply run the build target from the appropriate module e.g. to compile the broker source

2.11.2.3. Configuring Eclipse

1. Run the ant build from the root directory of Java trunk. 2. New project -> create from existing file system for broker, common, client, junit-toolkit, perftests, systests and each directory under management 4. Add the contents of lib/ to the build path 5. Setup Generated Code 6. Setup Dependencies

2.11.2.3.1. Generated Code

The Broker and Common packages both depend on generated code. After running 'ant' the build/scratch directory will contain this generated code. For the broker module add build/scratch/broker/src For the common module add build/scratch/common/src

2.11.2.3.2. Dependencies

These dependencies are correct at the time of writting however, if things are not working you can check the dependencies by looking in the modules build.xml file:

```
for i in `find . -name build.xml` ; do echo "$i:"; grep module.depends $i ; done
```

The *module.depend* value will detail which other modules are dependencies.

broker

- common
- management/common

client

- Common

systest

- client
- management/common
- broker
- broker/test
- common
- junit-toolkit
- management/tools/qpid-cli

perftests

- systests
- client
- broker
- common
- junit-toolkit

management/eclipse-plugin

- broker
- common
- management/common

management/console

- common
- client

management/agent

- common
- client

management/tools/qpid-cli

- common
- management/common

management/client

- common
- client

integrationtests

- systests
- client
- common
- junit-toolkit

testkit

- client
- broker
- common

tools

- client
- common

client/examples

- common
- client

broker-plugins

- client
- management/common
- broker
- common
- junit-toolkit

2.11.2.4. What next ?

If you want to run your built Qpid package, see our ??? for details of how to do that.

If you want to run our tests, you can use the ant test or testreport (produces a useful report) targets.

2.12. Other Queue Types

2.12.1. 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-, and last-value-queues (LVQs).

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 or programmatically.

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

2.12.2. 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.

2.12.3. 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.

2.12.4. 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".

2.12.5. Creating a Priority, Sorted or LVQ Queue

To create a priority, sorted or LVQ queue, it can be defined in the virtualhost configuration file, or the queue can be created programmtically from a client via AMQP (using an extension to JMS), or using JMX. 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.

2.12.5.1. Using configuration

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

2.12.5.1.1. Priority

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

Example 2.1. 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 2.2. Configuring a priority queue with fewer priorities

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

2.12.5.1.2. 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 2.3. Configuring a sorted queue

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

2.12.5.1.3. 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 2.4. 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 2.5. 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>
```

2.12.5.2. Using JMS 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 2.11.

Queue type	Argument name	Argument name	Argument Description
priority	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 2.6. 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");
((AMQSession<?,?>) session).createQueue(queueName, autoDelete, durable, exclusive,
```

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

Example 2.7. 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);
```

2.12.6. Low pre-fetch

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 or lvq 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.