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.Configuratio
2008-09-26 15:14:56,529 ERROR [main] server.Main (Main.java:206) - Error config
org.apache.commons.configuration.ConfigurationException: java.lang.reflect.Invo
at org.apache.qpid.server.security.auth.database.ConfigurationFilePrincipalData
at org.apache.qpid.server.security.auth.database.ConfigurationFilePrincipalData
at org.apache.qpid.server.security.auth.database.ConfigurationFilePrincipalData
at org.apache.qpid.server.registry.ConfigurationFileApplicationRegistry.initial
at org.apache.qpid.server.registry.ApplicationRegistry.initialise(ApplicationRe
at org.apache.qpid.server.registry.ApplicationRegistry.initialise(ApplicationRe
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
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
at java.lang.reflect.Method.invoke(Method.java:597)
at org.apache.qpid.server.security.auth.database.ConfigurationFilePrincipalData
```

.. 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 2.8, “ 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 Section 3.1.1.5, “ Qpid JMX Management Console User Guide ”

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.AMQMinaProtocolSessio
java.lang.NullPointerException
        at org.apache.qpid.server.security.access.PrincipalPermissions.authoris
        at org.apache.qpid.server.security.access.plugins.SimpleXML.authorise(S
        at org.apache.qpid.server.handler.QueueBindHandler.methodReceived(Queue
        at org.apache.qpid.server.handler.ServerMethodDispatcherImpl.dispatchQu
        at org.apache.qpid.framing.amqp_8_0.QueueBindBodyImpl.execute(QueueBind
        at org.apache.qpid.server.state.AMQStateManager.methodReceived(AMQState
        at org.apache.qpid.server.protocol.AMQMinaProtocolSession.methodFrameRe
        at org.apache.qpid.framing.AMQMethodBodyImpl.handle(AMQMethodBodyImpl.j
        at org.apache.qpid.server.protocol.AMQMinaProtocolSession.frameReceived
        at org.apache.qpid.server.protocol.AMQMinaProtocolSession.dataBlockRece
        at org.apache.qpid.server.protocol.AMQPFastProtocolHandler.messageRecei
        at org.apache.mina.common.support.AbstractIoFilterChain$TailFilter.mess
        at org.apache.mina.common.support.AbstractIoFilterChain.callNextMessage
        at org.apache.mina.common.support.AbstractIoFilterChain.access$1200(Abs
        at org.apache.mina.common.support.AbstractIoFilterChain$EntryImpl$1.mes
        at org.apache.qpid.pool.PoolingFilter.messageReceived(PoolingFilter.jav
        at org.apache.mina.filter.ReferenceCountingIoFilter.messageReceived(Ref
        at org.apache.mina.common.support.AbstractIoFilterChain.callNextMessage
        at org.apache.mina.common.support.AbstractIoFilterChain.access$1200(Abs
        at org.apache.mina.common.support.AbstractIoFilterChain$EntryImpl$1.mes
        at org.apache.mina.filter.codec.support.SimpleProtocolDecoderOutput.flu
        at org.apache.mina.filter.codec.QpidProtocolCodecFilter.messageReceived
        at org.apache.mina.common.support.AbstractIoFilterChain.callNextMessage
        at org.apache.mina.common.support.AbstractIoFilterChain.access$1200(Abs
        at org.apache.mina.common.support.AbstractIoFilterChain$EntryImpl$1.mes
        at org.apache.qpid.pool.Event$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$Worker.runTask(ThreadPoolExe
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecuto
        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$1.run()Ljava.lang.Object;
(URLClassLoader.java:194) at
jrockit.vm.AccessController.do_privileged_exc(Ljava.security.PrivilegedExceptionAction;Ljava.security.AccessContr
(Unknown Source) at
jrockit.vm.AccessController.doPrivileged(Ljava.security.PrivilegedExceptionAction;Ljava.security.AccessControlCon
(Unknown Source) at java.net.URLClassLoader.findClass(Ljava.lang.String;)Ljava.lang.Class;
(URLClassLoader.java:187) at
java.lang.ClassLoader.loadClass(Ljava.lang.String;Z)Ljava.lang.Class; (Unknown Source) at
sun.misc.Launcher$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 ca
        <flowResumeCapacity>8388608</flowResumeCapacity>  <!-- set the resume c
    </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
            <flowResumeCapacity>8388608</flowResumeCapacity>  <!-- set the resu
        </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 : S
```

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(71
```

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 : S
```

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(71
```

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
```

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.

1.5.2. Topic Configuration on Java Broker

New in 0.8 is the ability to define configuration for topics. Currently this is limited to configuration for slow consumer detection. This configuration is based on the work designed on the design wiki [http://cwiki.apache.org/confluence/display/qpid/Topic+Configuration+Design].

1.5.2.1. Topic Identification

A configuration section has two entries that can be used to identify how the configuration will be applied: 'name' and 'subscriptionName'.

```
<topic>
    <name>stocks.us</name>
```

```
<topic>
    <subscriptionName>clientid:mysubscription</subscriptio
```

It is also possible to combine these two identifiers to specify a unique subscription to a given topic.

```
<topic>
    <name>stocks.us</name>
    <subscriptionName>clientid:mysubscription</subscription
```

1.5.2.2. Configuration Items

Currently only one element of the designed configuration is processed, that of the slow consumer detection. This is setup as below using the 'slow-consumer-detection' element. There are two required types of tag, first the trigger, which is one of 'depth', 'messageAge' or 'messageCount' and secondly the 'policy'.

```
<slow-consumer-detection>
           <!-- The maximum depth before which the policy will be applied-
           <depth>4235264</depth>

           <!-- The maximum message age before which the policy will be ap
           <messageAge>600000</messageAge>

           <!-- The maximum number of message before which the policy will
           <messageCount>50</messageCount>

           <!-- Policy Selection -->
           <policy name="TopicDelete"/>
       </slow-consumer-detection>
```

The trigger is used to determine when the policy should be applied. Currently we have a simple policy 'topicdelete', this will disconnect consumers of topics where their consumption rate falls sufficiently to hit one of the trigger values.

1.5.2.3. Limitiations

As of 0.8 the topic configuration is limited to straight string matching. This means that given the following two topic configuring sections for 'stocks.us' and 'stocks.*' a subscription for 'stocks.uk' will not match the expected 'stocks.*'. Nor will any additional configuration listed in 'stocks.*' affect any 'stocks.us' subscriptions.

```
<topics>
    <topic>
     <name>stocks.us</name>
     ...
 </topic>
 <topic>
    <name>stocks.*</name>
    ...
    </topic>
</topics>
```

A subscription for 'stocks.us' will only receive configuration settings that are defined in the 'stocks.us' section.

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.

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.

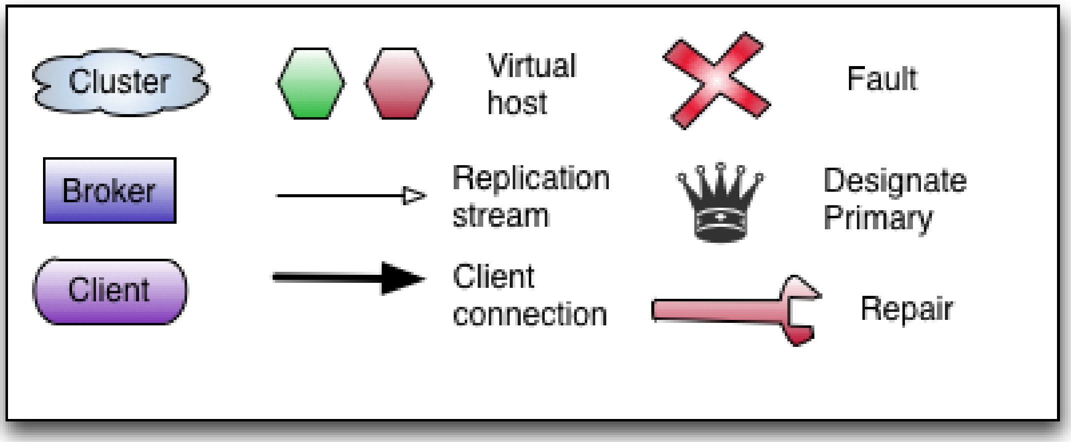
Clients connect to the cluster using a failover url. This allows the client to maintain a connection to the master in a way that is transparent to the client application.

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

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.

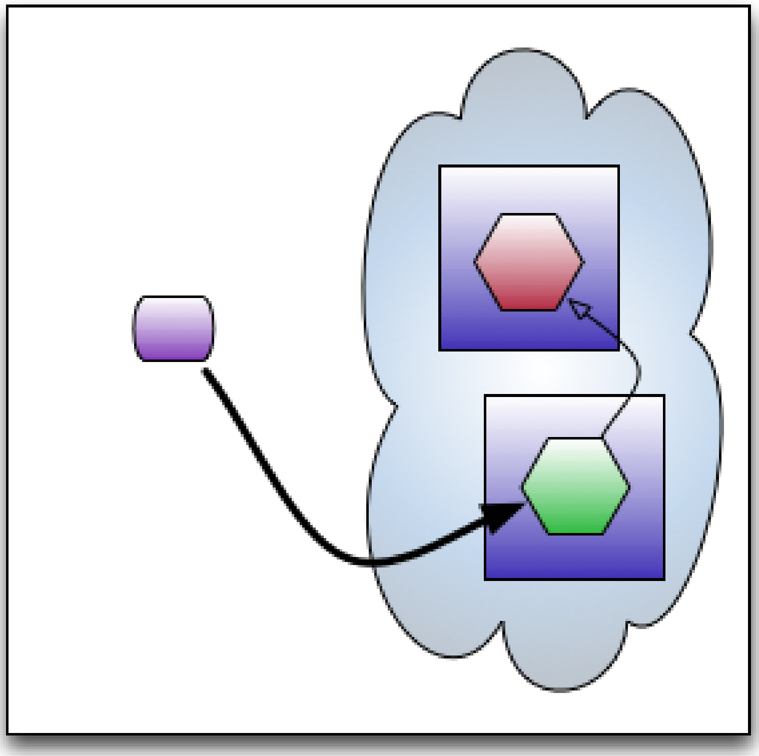
Figure 1.1. Key for figures



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

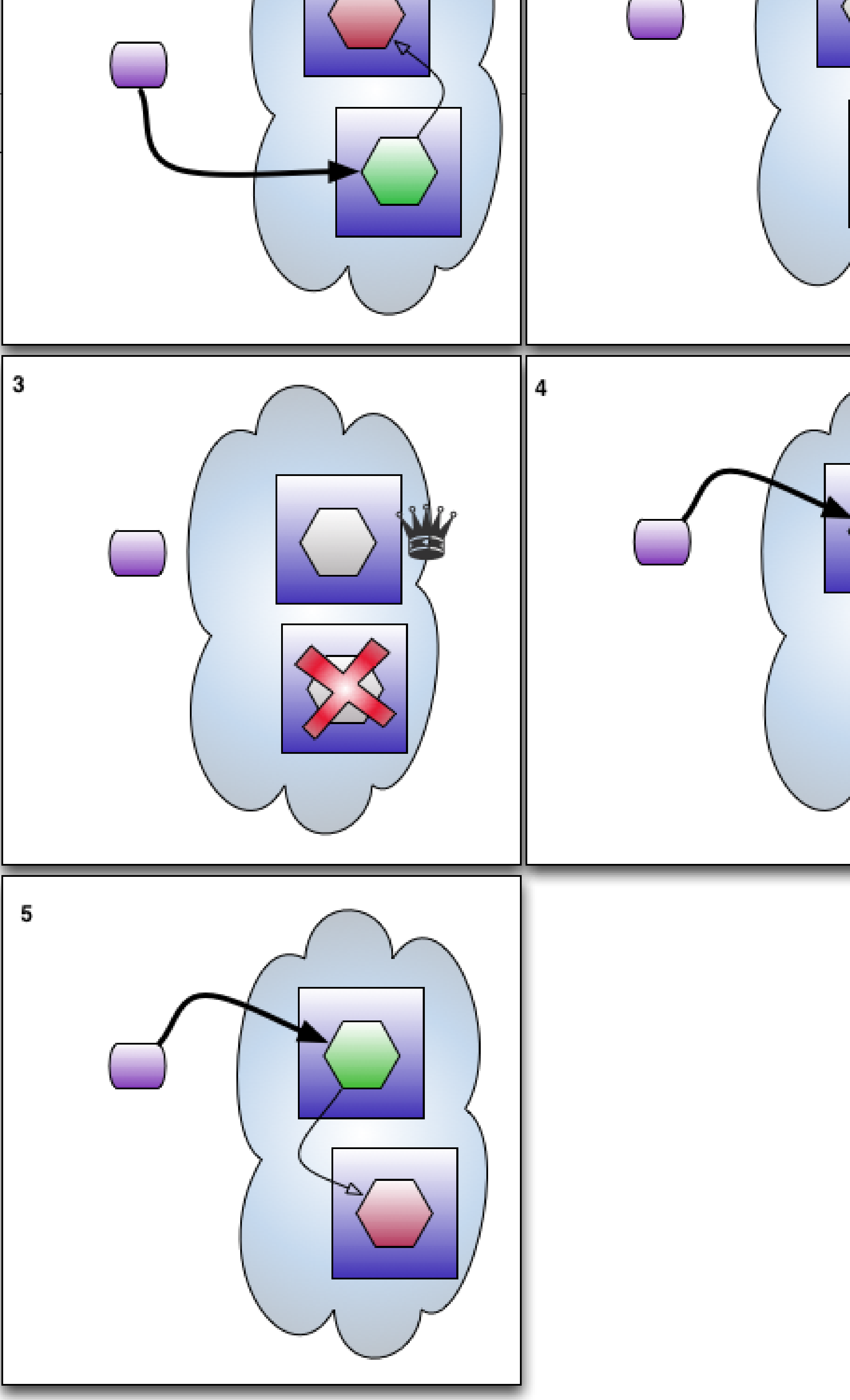
2. Master suffers a failure and disconnects all clients. Replica realises that it is no longer in contact with master. Clients begin to try to reconnect to the cluster, although these connection attempts will fail at this point.

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.

Figu



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.

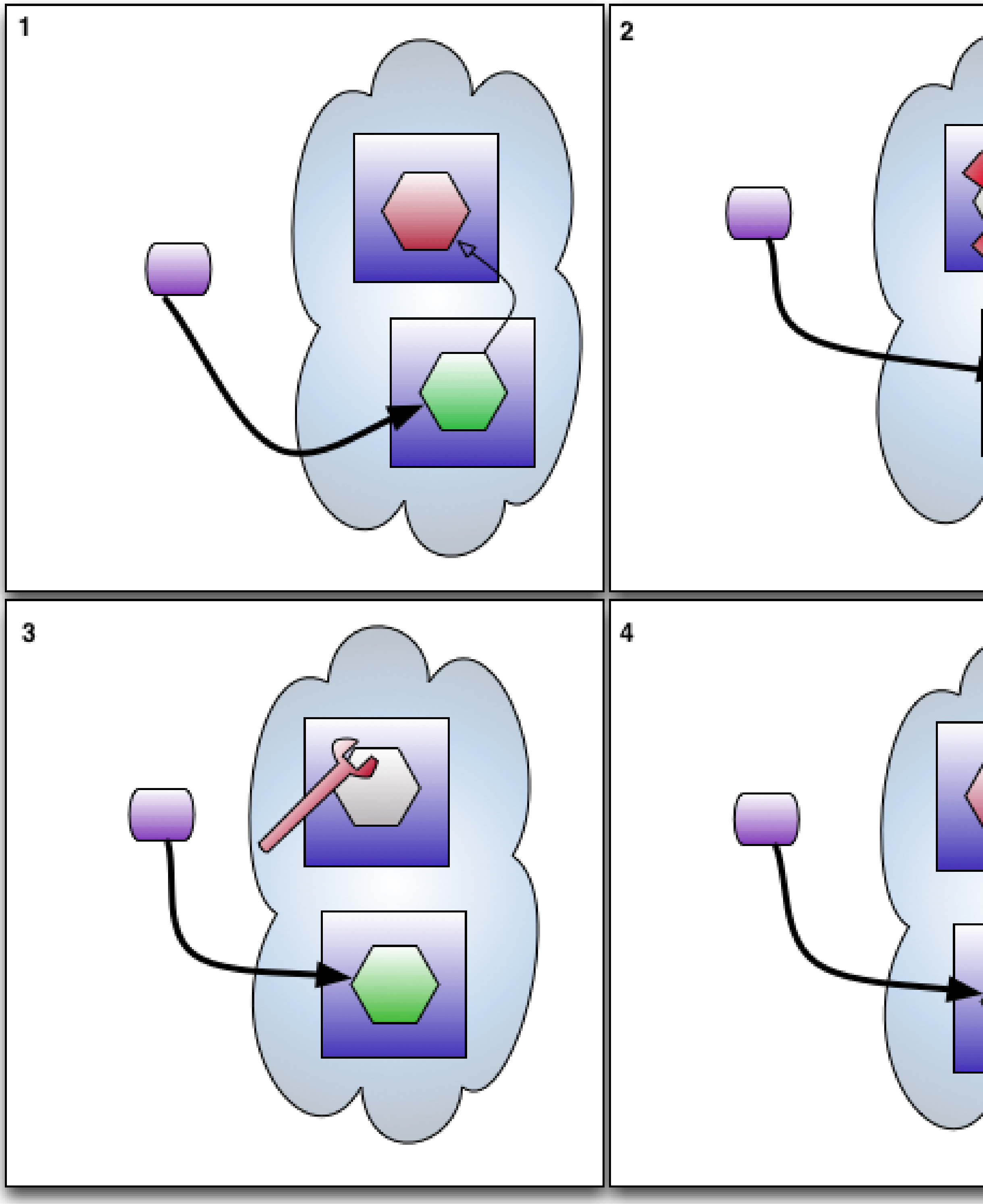
1. System operating normally

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.

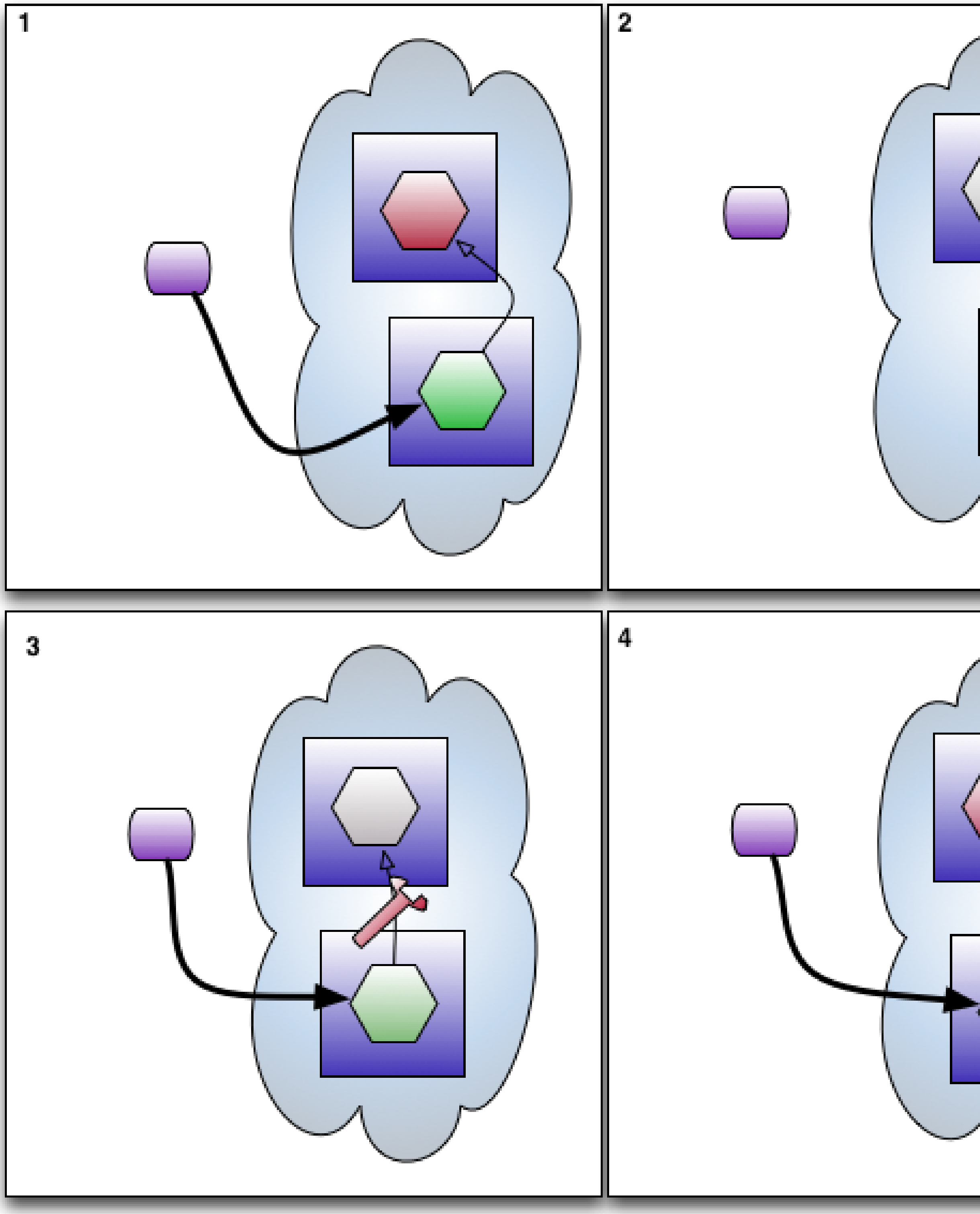
1. System operating normally

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.

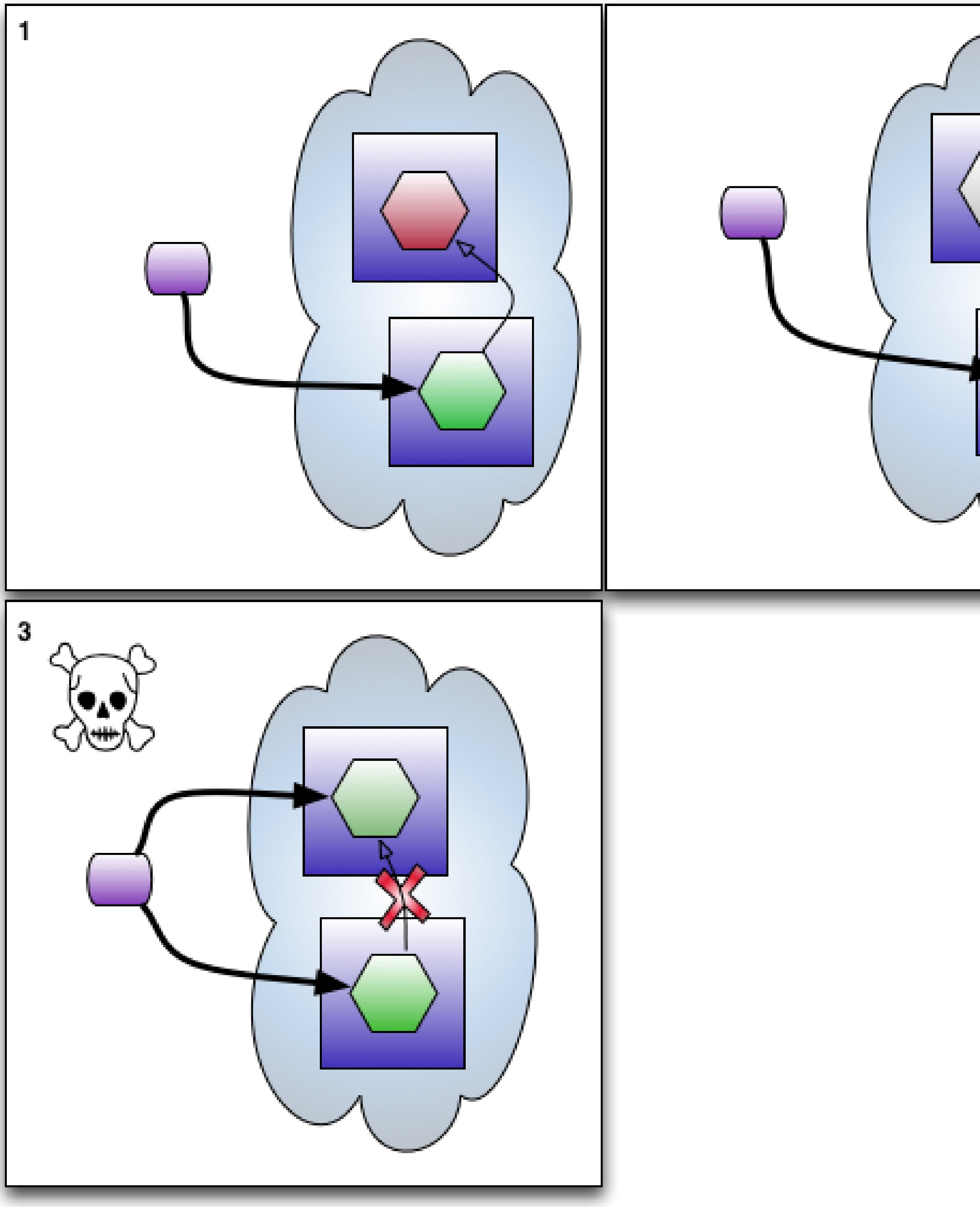
1. System operating normally

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.insufficientReplicasTimeout`.

```
       ...
    </highAvailability>
    ...
    <repConfig>
      <name>je.rep.insufficientReplicasTimeout</name>
      <value>2</value>
    </envConfig>
    <envConfig>
      <name>je.txn.timeout</name>
      <value>10 s</value>
    </envConfig>
    ...
  </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_FL
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.

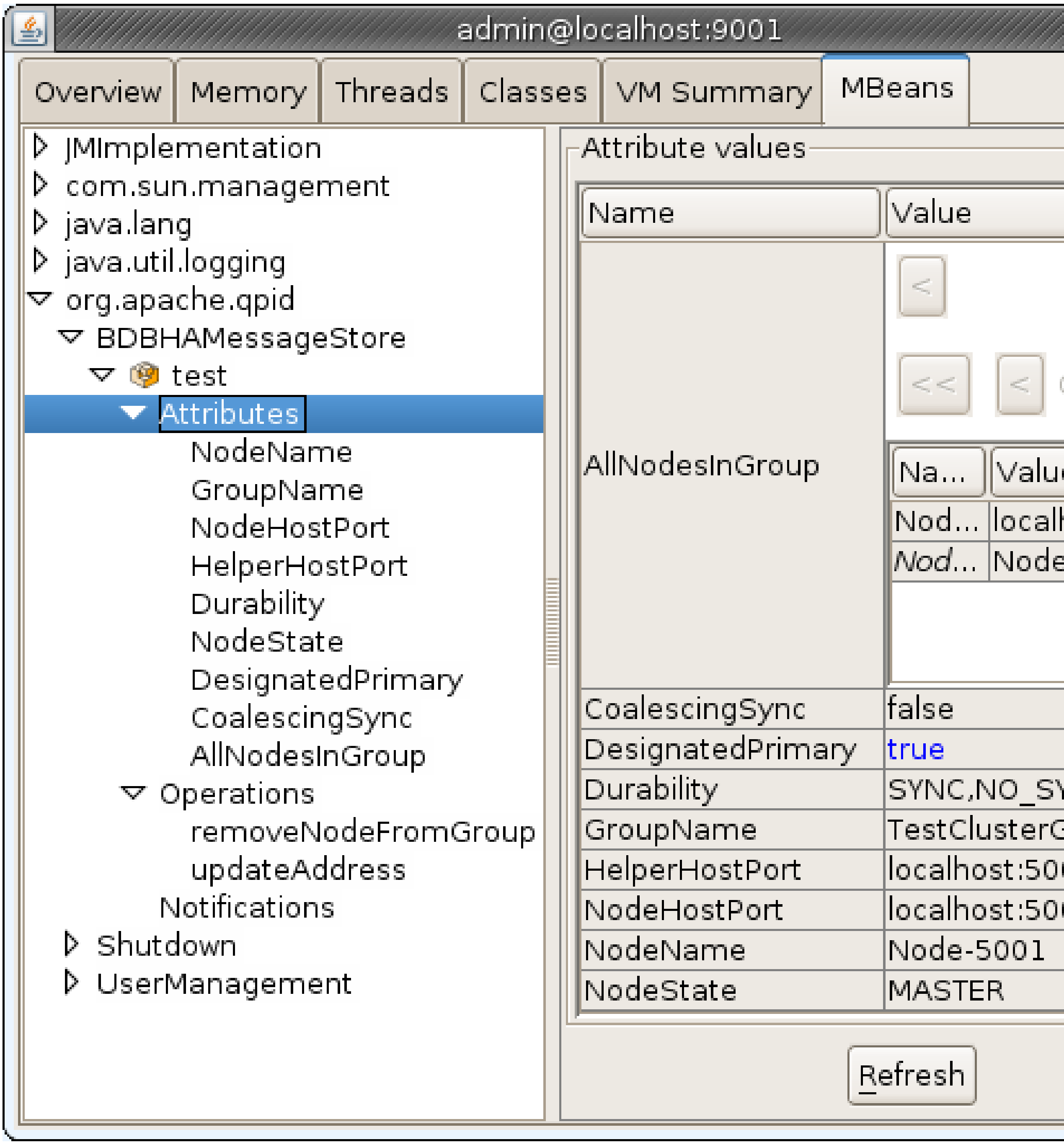
Table 1.3. Mbean `BDBHAMessageStore` attributes

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

Table 1.4. Mbean `BDBHAMessageStore` operations

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

Figure 1.7. BDBHAMessageStore view from jconsole.



Name	Value
AllNodesInGroup	
CoalescingSync	false
DesignatedPrimary	true
Durability	SYNC,NO_SY
GroupName	TestClusterG
HelperHostPort	localhost:50
NodeHostPort	localhost:50
NodeName	Node-5001
NodeState	MASTER

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
JMXConnector jmxConnector = JMXConnectorFactory.connect(url, environment);
MBeanServerConnection mbsc =  jmxConnector.getMBeanServerConnection();

ObjectName queueObjectName = new ObjectName("org.apache.qpid:type=BDBHAMessageS
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-5.0.48.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: 5.0.48
    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-5.0.48.jar DbEnableReplication -h /path/to/store -groupName MyReplicationGroup -nodeName MyNode1 -nodeHostPort localhost:5001
```

In the examples above, je jar of version 5.0.48 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-5.0.48.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 5.0.48 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.

Table 1.5. Number of producers/consumers in performance tests

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

The test was run against the following Qpid Broker configurations

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

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

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

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

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

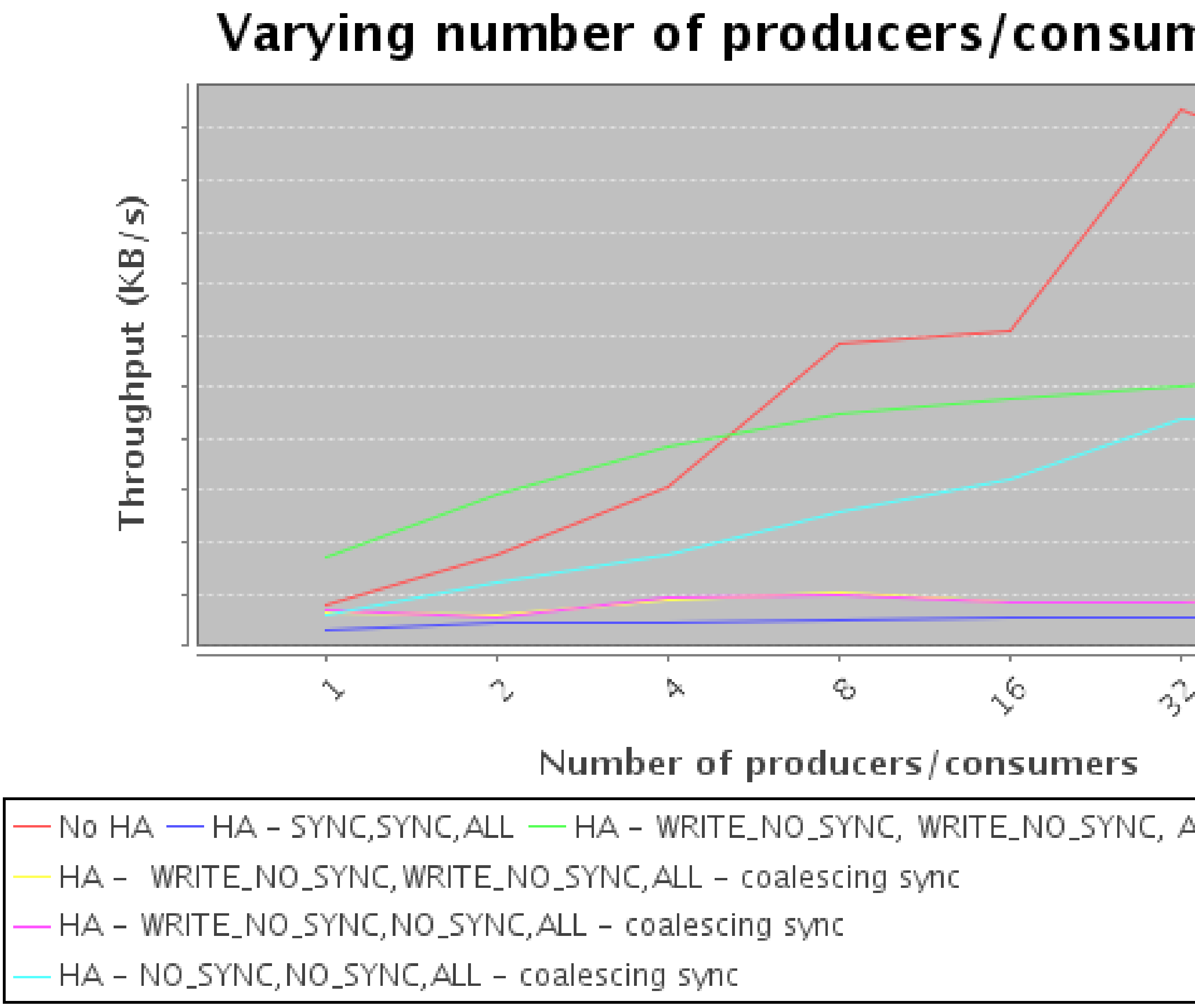
Each result is represented by throughput value in KB/second and difference in % between HA configuration and non HA case for the same number of clients.

Table 1.6. Performance Comparison

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

The figure below depicts the graphs for the performance test results

Figure 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.PlainPassw
                <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 Section 3.1.1.5, “ Qpid JMX Management Console User Guide ”)

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 Section 3.1.1.5, “ Qpid JMX Management Console User Guide ”).

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 Section 3.1.1.5, “ Qpid JMX Management Console User Guide ”)

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, ACLs specify which actions can be performed by each authenticated user. To enable the ACL <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/> 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 and group definitions. Each rule grants or denies specific rights to a user or group. Group definitions declare groups of users and serve to make the ACL file more concise.

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

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

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 a implict rule denying all operations to all users. It is as if each file ends with

```
ACL DENY ALL ALL
```

To allow all operations, other than those controlled by earlier use

```
ACL ALLOW ALL ALL
```

instead.

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, switch the DEBY-LOG to DENY for optimum performamce.

ACL rules are very powerful: it is possible to write very expressive rules permissioning every AMQP objects enumerating all object 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 (i.e. QUEUE) and apply consistently across the whole system.

2.2.3. Syntax

ACL rules must follow this syntax:

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

GROUP definitions must follow this syntax:

```
GROUP {group name} {username 1}..{username n} # Where username is a userna
```

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
GROUP securegroup bob \
alice # Another broker 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

QUEUE	A queue
EXCHANGE	An exchange
VIRTUALHOST	A virtualhost (Java Broker only)
METHOD	Management or agent or broker method (Java Broker only)
BROKER	The broker (not currently used in Java Broker)
LINK	A federation or inter-broker link (not currently used in Java Broker)

Table 2.5. ACL Rules:property

name	String. Object name, such as a queue name, exchange name or JMX method name.
durable	Boolean. Indicates the object is durable
routingkey	String. Specifies routing key
passive	Boolean. Indicates the presence of a *`passive`* flag
autodelete	Boolean. Indicates whether or not the object gets deleted when the connection is closed
exclusive	Boolean. Indicates the presence of an *`exclusive`* flag
temporary	Boolean. Indicates the presence of an *`temporary`* flag
type	String. Type of object, such as topic, fanout, or xml
alternate	String. Name of the alternate exchange
queuename	String. Name of the queue (used only when the object is something other than *`queue`*
component	String. JMX component name (Java Broker only)
schemapackage	String. QMF schema package name (Not used in Java Broker)
schemaclass	String. QMF schema class name (Not used in Java Broker)

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 three example ACLs illustrating some common use-cases.

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 allow to connect for messaging.

```
# Give operator permission to execute all JMX Methods
ACL ALLOW operator ALL METHOD
# Give operator permission to execute only read-only JMX Methods
ACL ALLOW readonly ACCESS METHOD
# Deny operator/readonly permission to perform messaging.
ACL DENY operator ACCESS VIRTUALHOST
ACL DENY readonly ACCESS VIRTUALHOST
...
... 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 a individual component and a group definition.

```
# Create a group usermaint with members bob and alice
GROUP usermaint bob alice
# Give operator permission to execute all JMX Methods
ACL ALLOW usermaint ALL METHOD component="UserManagement"
ACL DENY ALL ALL METHOD component="UserManagement"
...
... 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 required to create a temporary queue on which the server will resp
# 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

# Server side
# Allow the 'server' user to consume from the request queue and publish
# 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="TempQue

ACL DENY-LOG all all
```

2.3. Configure Java Qpid to use a SSL connection.

2.3.1. Using SSL connection with Qpid Java.

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

2.3.2. Setup

2.3.2.1. 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.

2.3.2.2. 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.

2.3.3. Performing the connection.

2.4. Configure Log4j CompositeRolling Appender

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

2.5. Configure the Broker via config.xml

2.5.1. 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 ???.

2.5.2. Qpid Version

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

??? ???

2.6. Configure the Virtual Hosts via virtualhosts.xml

2.6.1. 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.

2.6.1.1. 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 v
    <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 thi
            <!-- Always get amq.direct (for queues) and amq.topic (for topics) 
            <exchanges>
                <!-- Example of declaring an additional exchanges type for deve
                <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
                <!-- 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>
```

2.6.1.2. 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.

2.7. Debug using log4j

2.7.1. 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.

2.7.1.1. 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}
        <param name="Append" value="false"/>

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

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

</log4j:configuration>
```

2.7.1.2. 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}
        <param name="Append" value="false"/>

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

    <!-- 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
    <!-- 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>
```

2.8. How to Tune M3 Java Broker Performance

2.8.1. 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*.

2.8.2. 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.

2.8.2.1. 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.

2.8.3. 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.

2.9. Qpid Java Build How To

2.9.1. Build Instructions - General

2.9.1.1. Check out the source

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

???

2.9.1.2. 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.

2.9.2. 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.

2.9.2.1. 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

2.9.2.2. 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

2.9.2.3. 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

2.9.2.3.1. 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

2.9.2.3.2. 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 ; don
```

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

2.9.2.4. 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.

2.10. Use Priority Queues

2.10.1. General Information

The Qpid M3 release introduces priority queues into the Java Messaging Broker, supporting JMS clients who wish to make use of priorities in their messaging implementation.

There are some key points around the use of priority queues in Qpid, discussed in the sections below.

2.10.2. Defining Priority Queues

You must define a priority queue specifically before you start to use it. You cannot subsequently change a queue to/from a priority queue (without deleting it and re-creating).

You define a queue as a priority queue in the virtualhost configuration file, which the broker loads at startup. When defining the queue, add a <priority>true</priority> element. This will ensure that the queue has 10 distinct priorities, which is the number supported by JMS.

If you require fewer priorities, it is possible to specify a <priorities>int</priorities> element (where int is a valid 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".

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

2.10.3. Client configuration/messaging model for priority queues

There are some other configuration & paradigm changes which are required in order that priority queues work as expected.

2.10.3.1. Set low pre-fetch

Qpid clients receive buffered messages in batches, sized according to the pre-fetch value. The current default is 5000.

However, if you use the default value you will probably *not* see desirable behaviour with messages of different priority. This is because a message arriving after the pre-fetch buffer has filled will not leap frog messages of lower priority. It will be delivered at the front of the next batch of buffered messages (if that is appropriate), but this is most likely NOT what you need.

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.

Setting the Qpid pre-fetch to 1 for your client means that message priority will be honoured by the Qpid broker as it dispatches messages to your client. 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 ???

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

There is a slight performance cost here if using the receive() method and you could test with a slightly higher pre-fetch (up to 10) if the trade-off between throughput and prioritisation is weighted towards the former for your application. (If you're using OnMessage() then this is not a concern.)

2.10.3.2. Single consumer per session

If you are using the receive() method to consume messages then you should also only use one consumer per session with priority queues. If you're using OnMessage() then this is not a concern.

2.11. Slow Consumer Disconnect - User Guide

2.11.1. Introduction

Slow Consumer Disconnect (SCD) is a new feature in Qpid that provides a configurable mechanism to prevent a single slow consumer from causing a back up of unconsumed messages on the broker.

This is most relevant where Topics are in use, since a published message is not removed from the broker's memory until all subscribers have acknowledged that message.

Cases where a consumer is 'slow' can arise due to one of the following: poor network connectivity exists; a transient system issue affects a single client; a single subscriber written by a client team is behaving incorrectly and not acknowledging messages; a downstream resource such as a database is non-responsive.

SCD will enable the application owner to configure limits for a given consumer's queue and the behaviour to execute when those limits are reached.

2.11.2. What can it do?

SCD is only applicable to topics or durable subscriptions and can be configured on either a topic or a subscription name.

On triggering of a specified threshold the offending client will be disconnected from the broker with a 506 error code wrapped in a JMSException returned to the client via the ExceptionListener registered on the Connection object.

Note that it is essential that an ExceptionListener be specified by the client on creation of the connection and that exceptions coming back on that listener are handled correctly.

2.11.3. Frequency of SCD Checking

2.11.3.1. Configuring Frequency

You can configure the frequency with which the SCD process will check for slow consumers, along with the unit of time used to specify that frequency.

The *virtualhosts.virtualhost.hostname.slow-consumer-detection* elements *delay* and *timeunit* are used to specify the frequency and timeunit respectively in the virtualhosts.xml file e.g.

```
<virtualhosts>
 <default>test</default>
 <virtualhost>
  <name>test</name>
  <test>
     <slow-consumer-detection>
   <delay>60<delay/>
   <timeunit>seconds<timeunit/>
  <slow-consumer-detection/>
  </test>
 </virtualhost>
</virtualhosts>
```

2.11.3.2. SCD Log output

When the SCD component finds a queue with a configured threshold to check, the operational logging component (if enabled) will output the following line:

```
    SCD-1003 : Checking Status of Queue
```

2.11.4. Client Exceptions

When a Slow Consumer is disconnected, the client receives a 506 error from the broker wrapped in a JMSException and the Session and Connection are closed:

```
Dispatcher-Channel-1 2010-09-01 16:23:34,206 INFO [qpid.client.AMQSession.Dispa
    Dispatcher-Channel-1 thread terminating for channel 1:org.apache.qpid.clien
pool-2-thread-3 2010-09-01 16:23:34,238 INFO [apache.qpid.client.AMQConnection]
    :org.apache.qpid.AMQChannelClosedException: Error: Consuming to slow. [erro
javax.jms.JMSException: 506
at org.apache.qpid.client.AMQConnection.exceptionReceived(AMQConnection.java:13
at org.apache.qpid.client.protocol.AMQProtocolHandler.exception(AMQProtocolHand
at org.apache.qpid.client.protocol.AMQProtocolHandler.methodBodyReceived(AMQPro
at org.apache.qpid.client.protocol.AMQProtocolSession.methodFrameReceived(AMQPr
at org.apache.qpid.framing.AMQMethodBodyImpl.handle(AMQMethodBodyImpl.java:93)
at org.apache.qpid.client.protocol.AMQProtocolHandler$1.run(AMQProtocolHandler.
at org.apache.qpid.pool.Job.processAll(Job.java:110)
at org.apache.qpid.pool.Job.run(Job.java:149)
at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.ja
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:9
at java.lang.Thread.run(Thread.java:619)
Caused by: org.apache.qpid.AMQChannelClosedException: Error: Consuming to slow.
at org.apache.qpid.client.handler.ChannelCloseMethodHandler.methodReceived(Chan
at org.apache.qpid.client.handler.ClientMethodDispatcherImpl.dispatchChannelClo
at org.apache.qpid.framing.amqp_8_0.ChannelCloseBodyImpl.execute(ChannelCloseBo
at org.apache.qpid.client.state.AMQStateManager.methodReceived(AMQStateManager.
at org.apache.qpid.client.protocol.AMQProtocolHandler.methodBodyReceived(AMQPro
... 8 more
main 2010-09-01 16:23:34,316 INFO [apache.qpid.client.AMQSession] Closing sessi
    org.apache.qpid.client.AMQSession_0_8@ffeef1
```

2.11.5. Disconnection Thresholds

2.11.5.1. Topic Subscriptions

One key feature of SCD is the disconnection of a consuming client when a specified threshold is exceeded. For a pub-sub model using topics, this means that messages will no longer be delivered to the private queue which was associated with that consuming client, thus reducing any associated backlog in the broker.

2.11.5.2. Durable Topic Subscriptions

For durable subscriptions, simply disconnecting the consuming client will not suffice since the associated queue is by definition durable and messages would continue to flow to it after disconnection, potentially worsening any backing up of data on the broker.

The solution is to configure durable subscriptions to delete the underlying queue on disconnection. This means that messages will no longer be delivered to the private queue associated with the subscription, thus preventing any backlog.

Full details of how to configure the thresholds are provided below.

2.11.5.3. Message Age Threshold

You can configure SCD to be triggered on a topic or subscription when the oldest message in the associated private queue for the consumer ages beyond the specified value, in milliseconds.

2.11.5.4. Queue Depth Threshold

You can opt to use the depth of the queue in bytes as a threshold. SCD will be triggered by a queue depth greater than the threshold specified i.e. when a broker receives a message that takes the queue depth over the threshold.

2.11.5.5. Message Count Threshold

You can use the message count for the consumer's queue as the trigger, where a count higher than that specified will trigger disconnection.

2.11.5.6. Delete Policy

You can configure the policy you wish to apply in your broker configuration. There are currently 2 policies available:

Delete Temporary Queues Only

If you do not specify a <topicDelete/> element in your configuration, then only temporary queues associated with a topic subscription will be deleted on client disconnect. This is the default behaviour.

Delete Durable Subscription Queues

If you add the <topicDelete/> element with the sub-element <delete-persistent/> to your config, then the persistent queue which is associated with durable subscriptions to a topic will also be deleted. This is an important consideration since without deleting the underlying queue the client's unconsumed data will grow indefinitely while they will be unable to reconnect to that queue due to the SCD threshold configured, potentially having an adverse effect on the application or broker in use.

Example Topic Configuration

The following steps are required to configure SCD:

- Enable SCD checking for your virtual host
- Specify frequency for SCD checking
- Define thresholds for the topic
- Define the policy to apply on trigger

The example below shows a simple definition, with all three thresholds specified and a simple disconnection, with deletion of any temporary queue, defined.

For a durable subscription to this topic, no queue deletion would be applied on disconnect - which is likely to be undesirable (see section above).

```
<topics>
  <topic>
  <name>stocks.us.*</name>
   <slow-consumer-detection>
    <!-- The maximum depth before which -->
    <!-- the policy will be applied-->
    <depth>4235264</depth>
    <!-- The maximum message age before which -->
    <!-- the policy will be applied-->
    <messageAge>600000</messageAge>
    <!-- The maximum number of message before -->
    <!-- which the policy will be applied-->
    <messageCount>50</messageCount>
    <!-- Policy Selection -->
    <policy name="TopicDelete"/>
   </slow-consumer-detection>
  </topic>
</topics>
```

2.11.6. Important Points To Note

Client application developers should be educated about how to correctly handle being disconnected with a 506 error code, to avoid them getting into a thrashing state where they continually attempt to connect, fail to consume fast enough and are disconnected again.

Clients affected by slow consumer disconnect configuration should always use transactions where duplicate processing of an incoming message would have adverse affects, since they may receive a message more than once if disconnected before acknowledging a message in flight.

Chapter 3. Qpid JMX Management Console

3.1. Qpid JMX Management Console

3.1.1. Overview

The Qpid JMX Management Console is a standalone Eclipse RCP application that communicates with the broker using JMX.

3.1.1.1. Configuring Management Users

The Qpid Java broker has a single source of users for the system. So a user can connect to the broker to send messages and via the JMX console to check the state of the broker.

3.1.1.1.1. Adding a new management user

The broker does have some minimal configuration available to limit which users can connect to the JMX console and what they can do when they are there.

There are two steps required to add a new user with rights for the JMX console.

1. Create a new user login, see HowTo:???
2. Grant the new user permission to the JMX Console

3.1.1.1.1.1. Granting JMX Console Permissions

By default new users do not have access to the JMX console. The access to the console is controlled via the file *jmxremote.access*.

This file contains a mapping from user to privilege.

There are three privileges available:

1. readonly - The user is able to log in and view queues but not make any changes.
2. readwrite - Grants user ability to read and write queue attributes such as alerting values.
3. admin - Grants the user full access including ability to edit Users and JMX Permissions in addition to readwrite access.

This file is read at start up and can forcibly be reloaded by an admin user through the management console.

3.1.1.1.1.2. Access File Format

The file is a standard Java properties file and has the following format

```
<username>=<privilege>
```

If the username value is not a valid user (list in the specified PrincipalDatabase) then the broker will print a warning when it reads the file as that entry will have no meaning.

Only when the the username exists in both the access file and the PrincipalDatabase password file will the user be able to login via the JMX Console.

3.1.1.1.1.2.1. Example File

The file will be timestamped by the management console if edited through the console.

```
#Generated by JMX Console : Last edited by user:admin
#Tue Jun 12 16:46:39 BST 2007
admin=admin
guest=readonly
user=readwrite
```

3.1.1.2. Configuring Qpid JMX Management Console

3.1.1.2.1. Configuring Qpid JMX Management Console

Qpid has a JMX management interface that exposes a number of components of the running broker. You can find out more about the features exposed by the JMX interfaces ???.

3.1.1.2.1.1. Installing the Qpid JMX Management Console

1. Unzip the archive to a suitable location.

SSL encrypted connections

Recent versions of the broker can make use of SSL to encrypt their RMI based JMX connections. If a broker being connected to is making use of this ability then additional console configuration may be required, particularly when using self-signed certificates. See ??? for details.

JMXMP based connections

In previous releases of Qpid (M4 and below) the broker JMX connections could make use of the JMXMPConnector for additional security over its default RMI based JMX configuration. This is no longer the case, with SSL encrypted RMI being the favored approach going forward. However, if you wish to connect to an older broker using JMXMP the console will support this so long as the *jmxremote_optional.jar* file is provided to it. For details see ???.

3.1.1.2.1.2. Running the Qpid JMX Management Console

The console can be started in the following way, depending on platform:

- Windows: by running the 'qpidmc.exe' executable file.
- Linux: by running the 'qpidmc' executable.
- Mac OS X: by launching the consoles application bundle (.app file).

3.1.1.2.1.3. Using the Qpid JMX Management Console

Please see Section 3.1.1.5, “ Qpid JMX Management Console User Guide ” for details on using this Eclipse RCP application.

3.1.1.2.2. Using JConsole

See ???

3.1.1.2.3. Using HermesJMS

HermesJMS also offers integration with the Qpid management interfaces. You can get instructions and more information from HermesJMS [http://cwiki.apache.org/confluence/display/qpid/HermesJMS].

3.1.1.2.4. Using MC4J

MC4J [qpid_www.mc4j.org] is an alternative management tool. It provide a richer "dashboard" that can customise the raw MBeans.

3.1.1.2.4.1. Installation

- First download and install MC4J for your platform. Version 1.2 beta 9 is the latest version that has been tested.
- Copy the directory blaze/java/management/mc4j into the directory <MC4J-Installation>/dashboards

3.1.1.2.4.2. Configuration

You should create a connection the JVM to be managed. Using the Management->Create Server Connection menu option. The connection URL should be of the form: service:jmx:rmi:///jndi/rmi://localhost:8999/jmxrmi making the appropriate host and post changes.

3.1.1.2.4.3. Operation

You can view tabular summaries of the queues, exchanges and connections using the Global Dashboards->QPID tree view. To drill down on individual beans you can right click on the bean. This will show any available graphs too.

3.1.1.3. Management Console Security

3.1.1.3.1. Management Console Security

- Section 3.1.1.3.1.1, “ SSL encrypted RMI (0.5 and above) ”
- Section 3.1.1.3.1.2, “ JMXMP (M4 and previous) ”
- Section 3.1.1.3.1.3, “ User Accounts & Access Rights ”

3.1.1.3.1.1. SSL encrypted RMI (0.5 and above)

Current versions of the broker make use of SSL encryption to secure their RMI based JMX ConnectorServer for security purposes. This ships enabled by default, although the test SSL keystore used during development is not provided for security reasons (using this would provide no security as anyone could have access to it).

3.1.1.3.1.1.1. Broker Configuration

The broker configuration must be updated before the broker will start. This can be done either by disabling the SSL support, utilizing a purchased SSL certificate to create a keystore of your own, or generating a self-signed keystore.

The broker must be configured with a keystore containing the private and public keys associated with its SSL certificate. This is accomplished by setting the Java environment properties *javax.net.ssl.keyStore* and *javax.net.ssl.keyStorePassword* respectively with the location and password of an appropriate SSL keystore. Entries for these properties exist in the brokers main configuration file alongside the other management settings (see below), although the command line options will still work and take precedence over the configuration file.

```
<management>
    <ssl>
        <enabled>true</enabled>
        <!-- Update below path to your keystore location, eg ${conf}/qpid.keyst
        <keyStorePath>${conf}/qpid.keystore</keyStorePath>
        <keyStorePassword>password</keyStorePassword>
    </ssl>
</management>
```

3.1.1.3.1.1.2. JMX Management Console Configuration

If the broker makes use of an SSL certificate signed by a known signing CA (Certification Authority), the management console needs no extra configuration, and will make use of Java's built-in CA truststore for certificate verification (you may however have to update the system-wide default truststore if your CA is not already present in it).

If however you wish to use a self-signed SSL certificate, then the management console must be provided with an SSL truststore containing a record for the SSL certificate so that it is able to validate it when presented by the broker. This is performed by setting the *javax.net.ssl.trustStore* and *javax.net.ssl.trustStorePassword* environment variables when starting the console. This can be done at the command line, or alternatively an example configuration has been made within the console's qpidmc.ini launcher configuration file that may pre-configured in advance for repeated usage. See the Section 3.1.1.5, “ Qpid JMX Management Console User Guide ” for more information on this configuration process.

3.1.1.3.1.1.3. JConsole Configuration

As with the JMX Management Console above, if the broker is using a self-signed SSL certificate then in order to connect remotely using JConsole, an appropriate trust store must be provided at startup. See ??? for further details on configuration.

3.1.1.3.1.1.4. Additional Information

More information on Java's handling of SSL certificate verification and customizing the keystores can be found in the http://java.sun.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html#CustomizingStores.

3.1.1.3.1.2. JMXMP (M4 and previous)

In previous releases of Qpid (M4 and below) the broker, can make use of Sun's Java Management Extensions Messaging Protocol (JMXMP) to provide encryption of the JMX connection, offering increased security over the default unencrypted RMI based JMX connection.

3.1.1.3.1.2.1. Download and Install

This is possible by adding the jmxremote_optional.jar as provided by Sun. This jar is covered by the Sun Binary Code License and is not compatible with the Apache License which is why this component is not bundled with Qpid.

Download the JMX Remote API 1.0.1_04 Reference Implementation from ???. The included 'jmxremote-1_0_1-bin\lib\jmxremote_optional.jar' file must be added to the broker classpath:

First set your classpath to something like this:

```
CLASSPATH=jmxremote_optional.jar
```

Then, run qpid-server passing the following additional flag:

```
qpid-server -run:external-classpath=first
```

Following this the configuration option can be updated to enabled use of the JMXMP based JMXConnectorServer.

3.1.1.3.1.2.2. Broker Configuration

To enabled this security option change the *security-enabled* value in your broker configuration file.

```
<management>
    <security-enabled>true</security-enabled>
</management>
```

You may also (for M2 and earlier) need to set the following system properties using the environment variable QPID_OPTS:

QPID_OPTS="-Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=8999 -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false"

3.1.1.3.1.2.3. JMX Management Console Configuration

If you wish to connect to a broker configured to use JMXMP then the console also requires provision of the Optional sections of the JMX Remote API that are not included within the JavaSE platform.

In order to make it available to the console, place the 'jmxremote_optional.jar' (rename the file if any additional information is present in the file name) jar file within the 'plugins/jmxremote.sasl_1.0.1/' folder of the console release (on Mac OS X you will need to select 'Show package contents' from the context menu whilst selecting the management console bundle in order to reveal the inner file tree).

Following the the console will automatically load the JMX Remote Optional classes and attempt the JMXMP connection when connecting to a JMXMP enabled broker.

3.1.1.3.1.3. User Accounts & Access Rights

In order to access the management operations via JMX, users must have an account and have been assigned appropriate access rights. See ???

3.1.1.4. Qpid JMX Management Console FAQ

3.1.1.4.1. Errors

3.1.1.4.1.1. How do I connect the management console to my broker using security ?

The ??? page will give you the instructions that you should use to set this up.

3.1.1.4.1.2. I am unable to connect Qpid JMX MC/JConsole to a remote broker running on Linux, but connecting to localhost on that machine works ?

The RMI based JMX ConnectorServer used by the broker requries two ports to operate. The console connects to an RMI Registry running on the primary (default 8999) port and retrieves the information actually needed to connect to the JMX Server. This information embeds the hostname of the remote machine, and if this is incorrect or unreachable by the connecting client the connection will fail.

This situation arises due to the hostname configuration on Linux and is generally encountered when the remote machine does not have a DNS hostname entry on the local network, causing the hostname command to return a loopback IP instead of a fully qualified domain name or IP address accessible by remote client machines. It is described in further detail at: ???

To remedy this issue you can set the *java.rmi.server.hostname* system property to control the hostname/ip reported to the RMI runtime when advertising the JMX ConnectorServer. This can also be used to dictate the address returned on a computer with multiple network interfaces to control reachability. To do so, add the value *-Djava.rmi.server.hostname=<desired hostname/ip>* to the QPID_OPTS environment variable before starting the *qpid-server* script.

3.1.1.5. Qpid JMX Management Console User Guide

3.1.1.5.1. Qpid JMX Management Console User Guide

The Qpid JMX Management Console is a standalone Eclipse RCP application for managing and monitoring the Qpid Java server utilising its JMX management interfaces.

This guide will give an overview of configuring the console, the features supported by it, and how to make use of the console in managing the various JMX Management Beans (MBeans) offered by the Qpid Java server.

3.1.1.5.2. Startup & Configuration

3.1.1.5.2.1. Startup

The console can be started in the following way, depending on platform:

- *Windows:* by running the *qpidmc.exe* executable file.
- *Linux:* by running the *qpidmc* executable.
- *Mac OS X:* by launching the *Qpid Management Console.app* application bundle.

3.1.1.5.2.2. SSL configuration

Newer Qpid Java servers can protect their JMX connections with SSL, and this is enabled by default. When attempting to connect to a server with this enabled, the console must be able to verify the SSL certificate presented to it by the server or the connection will fail.

If the server makes use of an SSL certificate signed by a known Signing CA (Certification Authority) then the console needs no extra configuration, and will make use of Java's default system-wide CA TrustStore for certificate verification (you may however have to update the system-wide default CA TrustStore if your certified is signed by a less common CA that is not already present in it).

If however the server is equipped with a self-signed SSL certificate, then the management console must be provided with an appropriate SSL TrustStore containing the public key for the SSL certificate, so that it is able to validate it when presented by the server. The server ships with a script to create an example self-signed SSL certificate, and store the relevant entries in a KeyStore and matching TrustStore. This script can serve as a guide on how to use the Java Keytool security utility to manipulate your own stores, and more information can be found in the JSSE Reference Guide: http://java.sun.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html#CustomizingStores.

Supplying the necessary details to the console is performed by setting the *javax.net.ssl.trustStore* and *javax.net.ssl.trustStorePassword* environment variables when starting it. This can be done at the command line, but the preferred option is to set the configuration within the *qpidmc.ini* launcher configuration file for repeated usage. This file is equipped with a template to ease configuration, this should be uncommented and edited to suit your needs. It can be found in the root of the console releases for Windows, and Linux. For Mac OS X the file is located within the consoles *.app* application bundle, and to locate and edit it you must select *'Show Package Contents'* when accessing the context menu of the application, then browse to the *Contents/MacOS* sub folder to locate the file.

3.1.1.5.2.3. JMXMP configuration

Older releases of the Qpid Java server can make use of the Java Management Extensions Messaging Protocol (JMXMP) to provide protection for their JMX connections. This occurs when the server has its main configuration set with the management *'security-enabled'* property set to true.

In order to connect to this configuration of server, the console needs an additional library that is not included within the Java SE platform and cannot be distributed with the console due to licensing restrictions.

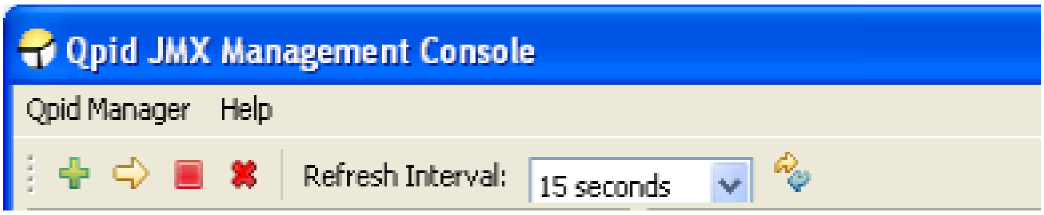
You can download the JMX Remote API 1.0.1_04 Reference Implementation from the Sun website ???. The included *jmxremote-1_0_1-bin/lib/jmxremote_optional.jar* file must be added to the *plugins/jmxremote.sasl_1.0.1* folder of the console release (again, in Mac OS X you will need to select *'Show* *package contents'* from the context menu whilst selecting the management console bundle in order to reveal the inner file tree).

Following this the console will automatically load the JMX Remote Optional classes and negotiate the SASL authentication profile type when encountering a JMXMP enabled Qpid Java server.

3.1.1.5.3. Managing Server Connections

3.1.1.5.3.1. Main Toolbar

The main toolbar of the console can be seen in the image below. The left most buttons respectively allow for adding a new server connection, reconnecting to an existing server selected in the connection tree, disconnecting the selected server connection, and removing the server from the connection tree.

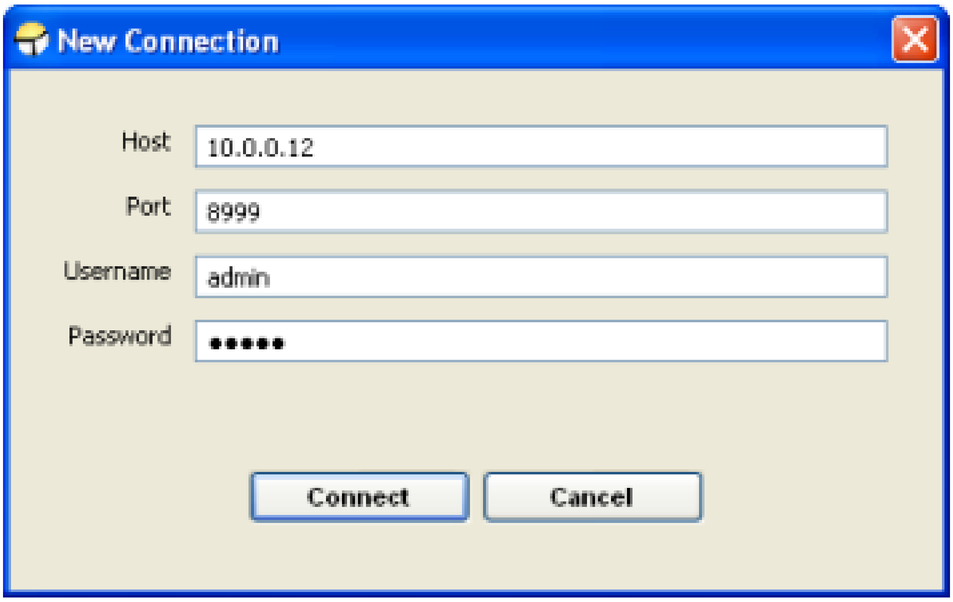


Refresh Interval: 15 seconds

Beside these buttons is a combo for selecting the refresh interval; that is, how often the console requests updated information to display for the currently open area in the main view. Finally, the right-most button enables an immediate update.

3.1.1.5.3.2. Connecting to a new server

To connect to a new server, press the *Add New Server* toolbar button, or select the *Qpid Manager -> Add New Connection* menu item. At this point a dialog box will be displayed requesting the server details, namely the server hostname, management port, and a username and password. An example is shown below:



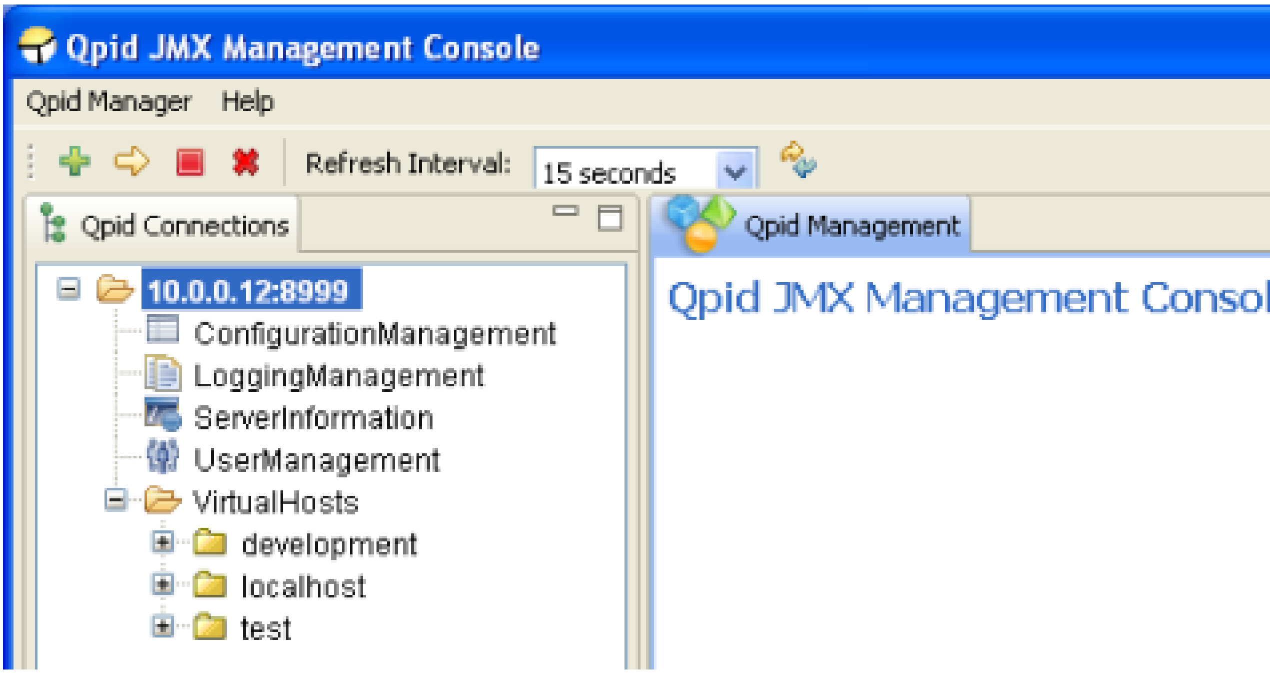
Host 10.0.0.12

Port 8999

Username admin

Password •••••

Once all the required details are entered, pressing Connect will initiate a connection attempt to the server. It the attempt fails a reason will be shown and the server will not be added to the connection tree. If the attempt is successful the server will be added to the connections list and the entry expanded to show the initial administration MBeans the user has access to and any VirtualHosts present on the server, as can be seen in the figure below.



If the server supports a newer management API than the console in use, once connected this initial screen will contain a message on the right, indicating an upgraded console should be sought by the user to ensure all management functionality supported by the server is being utilised.

3.1.1.5.3.3. Reconnecting to a server

If a server has been connected to previously, it will be saved as an entry in the connection tree for further use. On subsequent connections the server can simply be selected from the tree and using the *Reconnect* toolbar button or *Qpid Manager -> Reconnect* menu item. At this stage the console will prompt simply for the username and password with which the user wishes to connect, and following a successful connection the screen will appear as shown previously above.

3.1.1.5.3.4. Disconnecting from a server

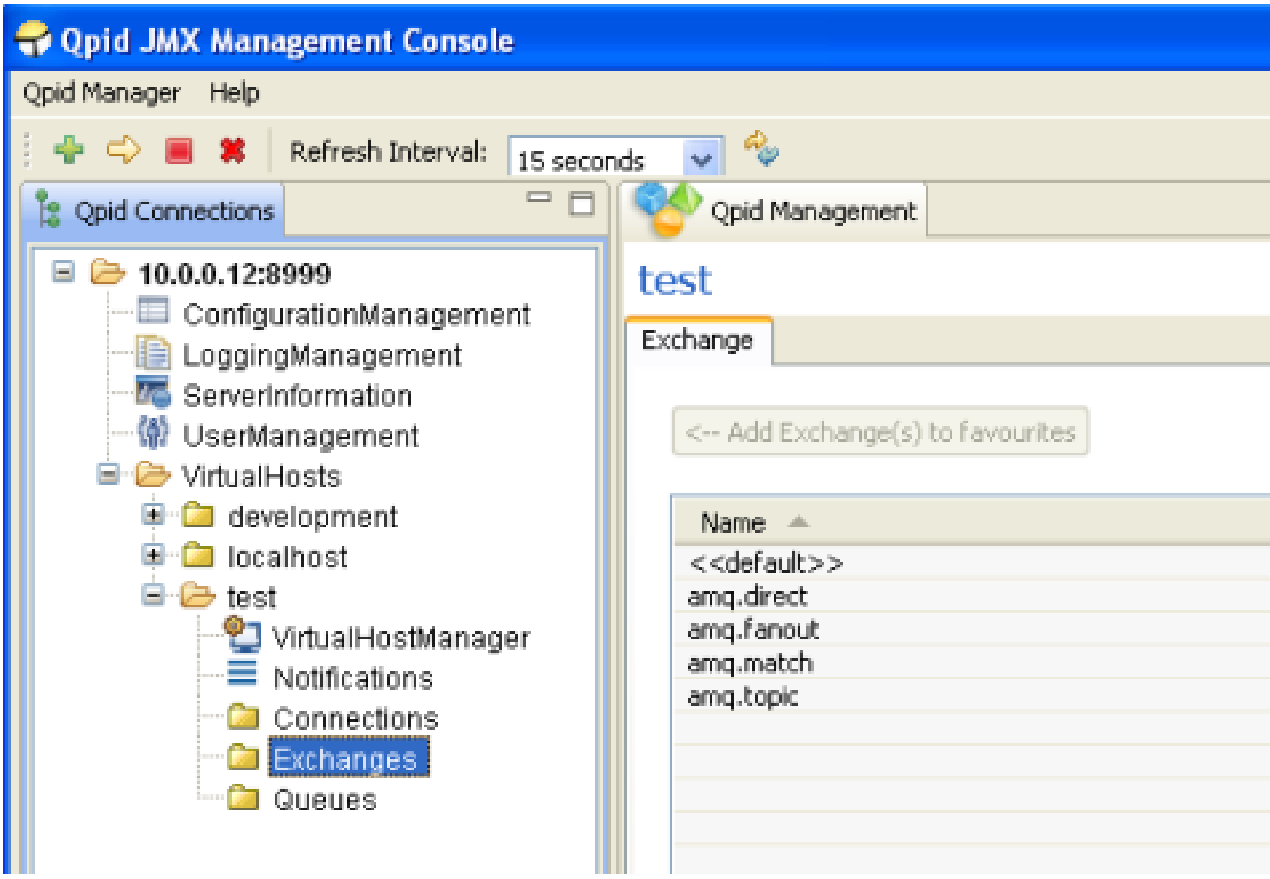
To disconnect from a server, select the connection tree node for the server and press the *Disconnect* toolbar button, or use the *Qpid Manager -> Disconnect* menu option.

3.1.1.5.3.5. Removing a server

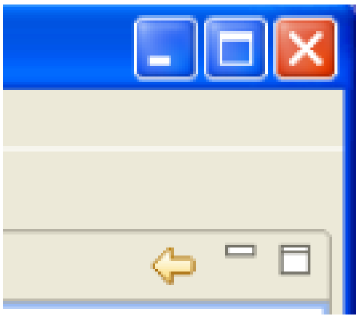
To remove a server from the connection list, select the connection tree node for the server and press the *Remove* toolbar button, or use the *Qpid Manager -> Remove Connection* menu option.

3.1.1.5.4. Navigating a connected server

Once connected to a server, the various areas available for administration are accessed using the Qpid Connections tree at the left side of the application. To open a particular MBean from the tree for viewing, simply select it in the tree and it will be opened in the main view.



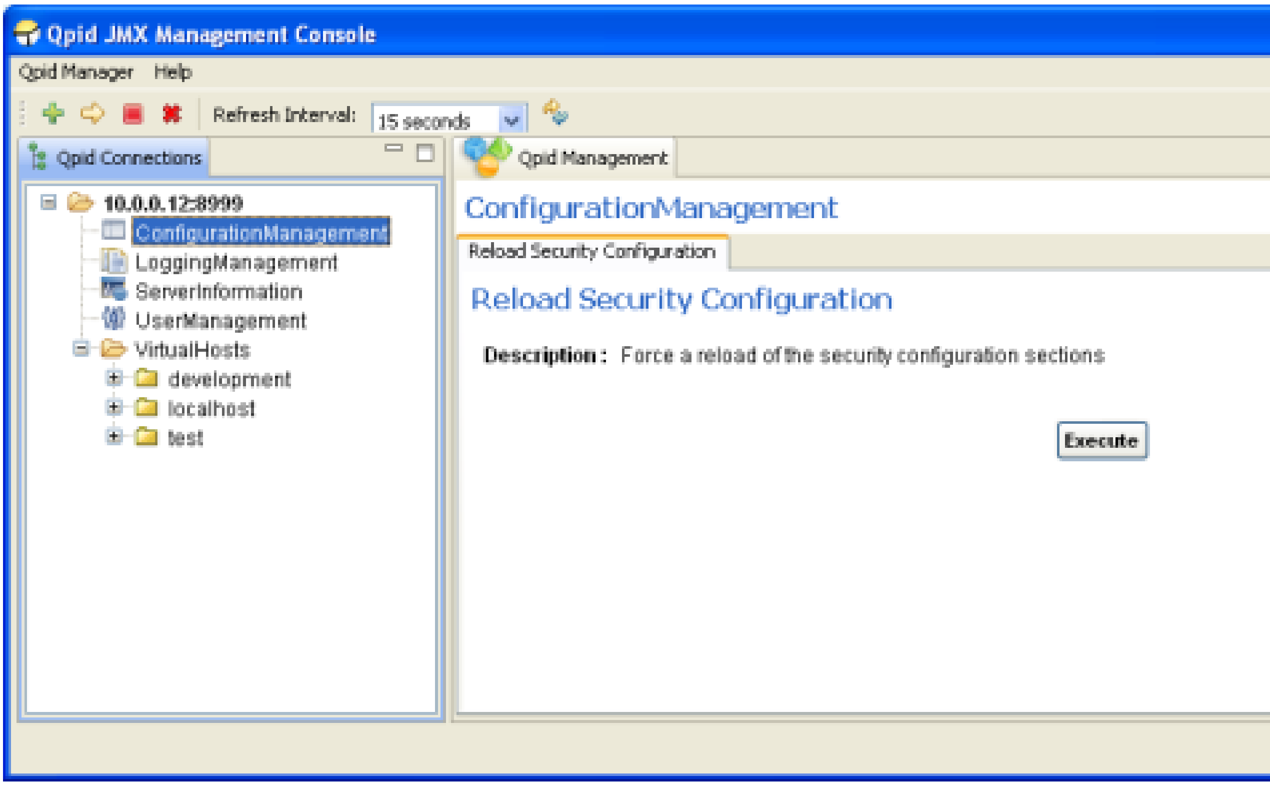
As there may be vast numbers of Queues, Connections, and Exchanges on the server these MBeans are not automatically added to the tree along with the general administration MBeans. Instead, dedicated selection areas are provided to allow users to select which Queue/Connection/Exchange they wish to view or add to the tree. These areas can be found by clicking on the Connections, Exchanges, and Queues nodes in the tree under each VirtualHost, as shown in the figure above. One or more MBeans may be selected and added to the tree as Favourites using the button provided. These settings are saved for future use, and each time the console connects to the server it will check for the presence of the MBean previously in the tree and add them if they are still present. Queue/Connection/Exchange MBeans can be removed from the tree by right clicking on them to expose a context menu allowing deletion.



As an alternative way to open a particular MBean for viewing, without first adding it to the tree, you can simply double click an entry in the table within the Queue/Connection/Exchange selection areas to open it immediately. It is also possible to open some MBeans like this whilst viewing certain other MBeans. When opening an MBean in either of these ways, a Back button is enabled in the top right corner of the main view. Using this button will return you to the selection area or MBean you were previously viewing. The history resets each time the tree is used to open a new area or MBean.

3.1.1.5.5. ConfigurationManagement MBean

The ConfigurationManagement MBean is available on newer servers, to users with admin level management rights. It offers the ability to perform a live reload of the *Security* sections defined in the main server configuration file (e.g. defaults to: *etc/config.xml*). This is mainly to allow updating the server Firewall configuration to new settings without a restart, and can be performed by clicking the Execute button and confirming the prompt which follows.

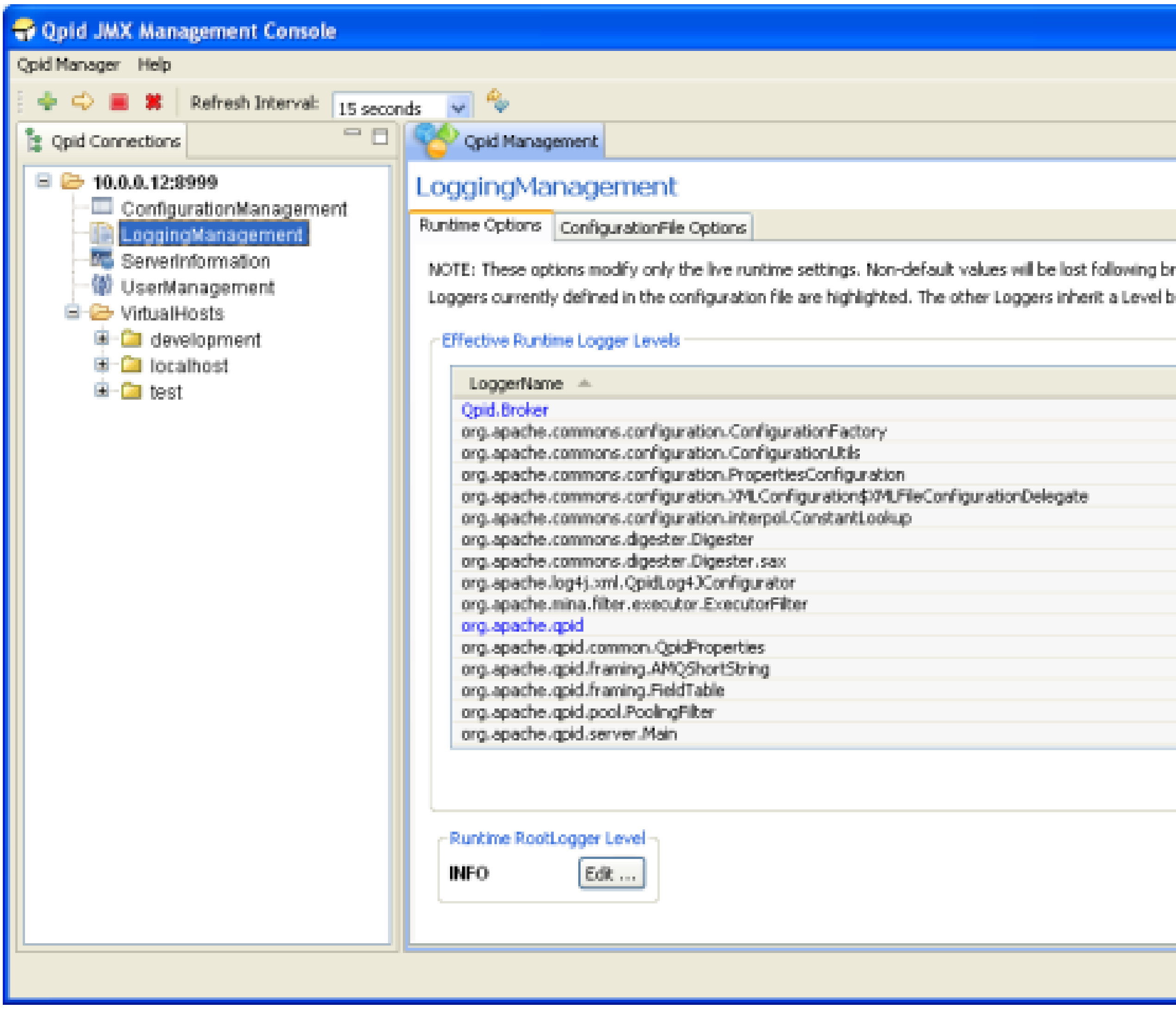


Description : Force a reload of the security configuration sections

3.1.1.5.6. LoggingManagement MBean

The LoggingManagement MBean is available on newer servers, and accessible by admin level users. It allows live alteration of the logging behaviour, both at a Runtime-only level and at the configuration file level. The latter can optionally affect the Runtime configuration, either through use of the servers automated LogWatch ability which detects changes to the configuration file and reloads it, or by manually requesting a reload. This functionality is split across two management tabs, Runtime Options and ConfigurationFile Options.

3.1.1.5.6.1. Runtime Options



NOTE: These options modify only the live runtime settings. Non-default values will be lost following br

Loggers currently defined in the configuration file are highlighted. The other Loggers inherit a Level b

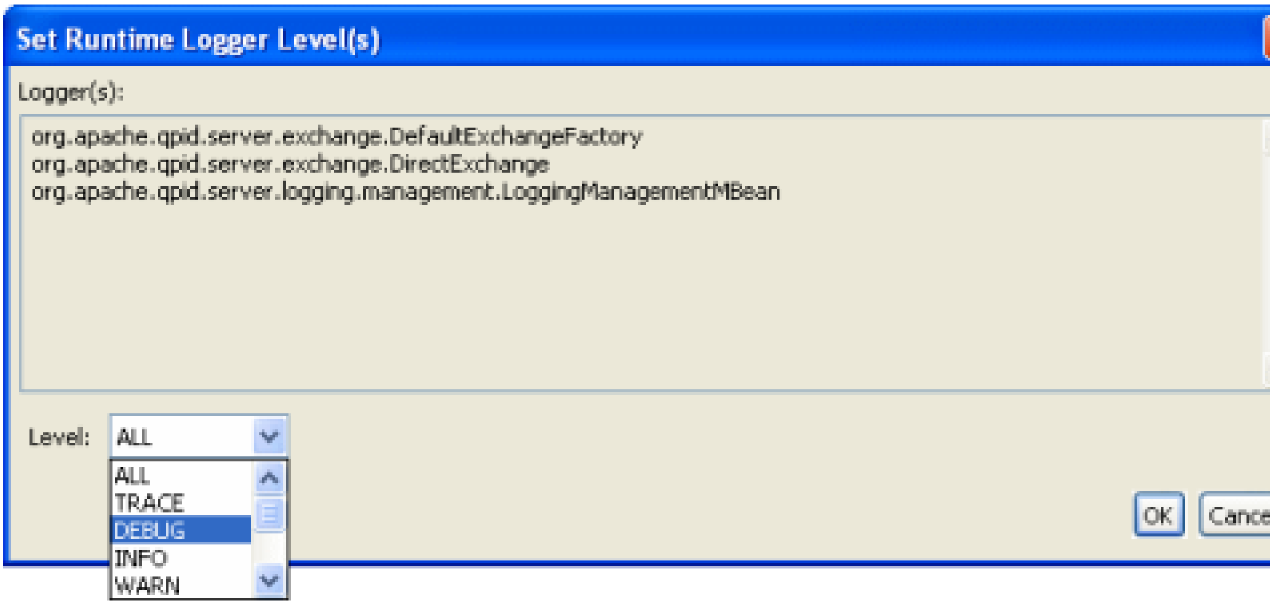
LoggerName
Qpid.Broker
org.apache.commons.configuration.ConfigurationFactory
org.apache.commons.configuration.ConfigurationUtils
org.apache.commons.configuration.PropertiesConfiguration
org.apache.commons.configuration.XMLConfiguration$XMLFileConfigurationDelegate
org.apache.commons.configuration.interpol.ConstantLookup
org.apache.commons.digester.Digester
org.apache.commons.digester.Digester.sax
org.apache.log4j.xml.QpidLog4JConfigurator
org.apache.mina.filter.executor.ExecutorFilter
org.apache.qpid
org.apache.qpid.common.QpidProperties
org.apache.qpid.framing.AMQShortString
org.apache.qpid.framing.FieldTable
org.apache.qpid.pool.PoolingFilter
org.apache.qpid.server.Main

INFO

The Runtime Options tab allows manipulation of the logging settings without affecting the configuration files (this means the changes will be lost when the server restarts), and gives individual access to every Logger active within the server.

As shown in the figure above, the table in this tab presents the Effective Level of each Logger. This is because the Loggers form a hierarchy in which those without an explicitly defined (in the logging configuration file) Level will inherit the Level of their immediate parent; that is, the Logger whose full name is a prefix of their own, or if none satisfy that condition then the RootLogger is their parent. As example, take the *org.apache.qpid* Logger. It is parent to all those below it which begin with *org.apache.qpid* and unless they have a specific Level of their own, they will inherit its Level. This can be seen in the figure, whereby all the children Loggers visible have a level of WARN just like their parent, but the RootLogger Level is INFO; the children have inherited the WARN level from *org.apache.qpid* rather than INFO from the RootLogger.

To aid with this distinction, the Logger Levels that are currently defined in the configuration file are highlighted in the List. Changing these levels at runtime will also change the Level of all their children which haven't been set their own Level using the runtime options. In the latest versions of the LoggingManagement MBean, it is possible to restore a child logger that has had an explicit level se, to inheriting that of its parent by setting it to an INHERITED level that removes any previously set Level of its own.



Set Runtime Logger Level(s)

Logger(s):

```
org.apache.qpid.server.exchange.DefaultExchangeFactory
org.apache.qpid.server.exchange.DirectExchange
org.apache.qpid.server.logging.management.LoggingManagementMBean
```

Level: ALL

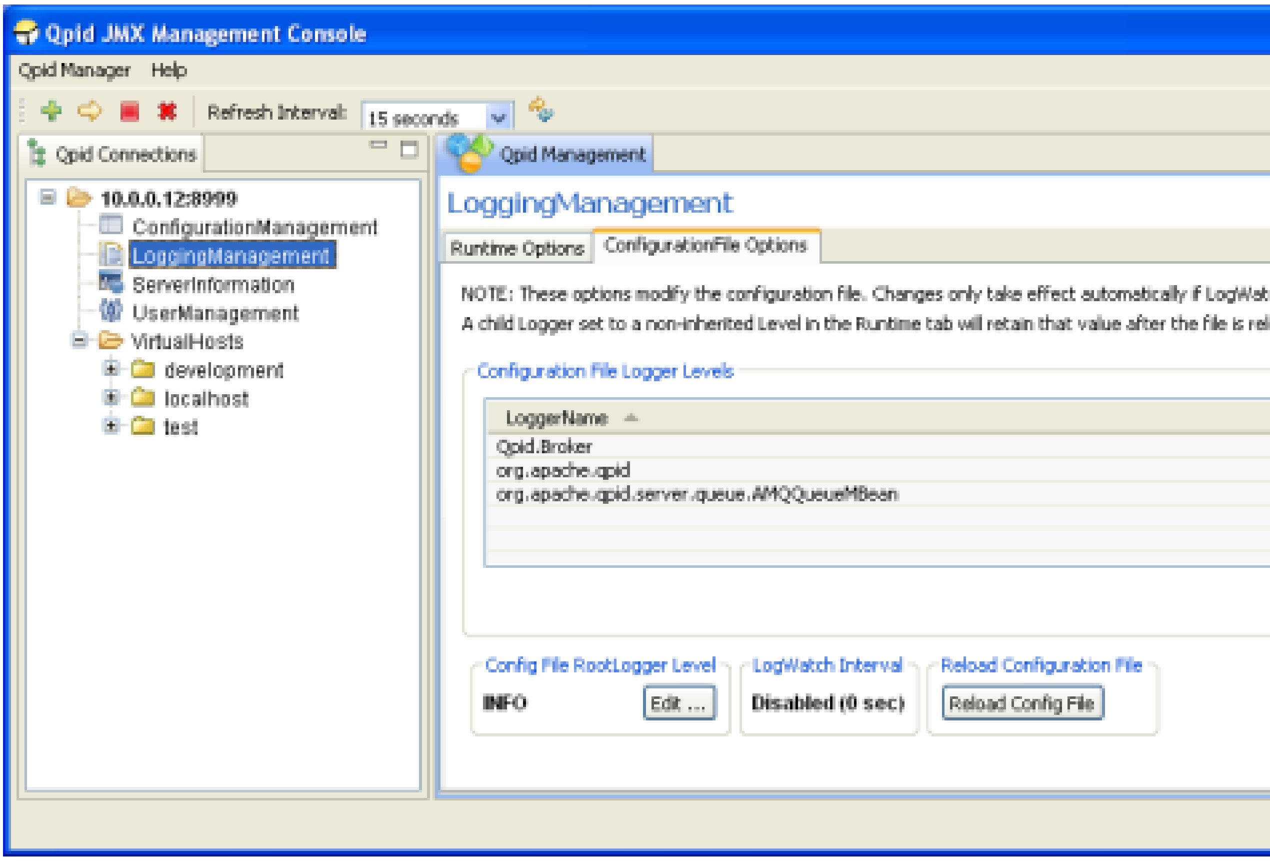
In order to set one of more Loggers to a new Level, they should be selected in the table (or double click an individual Logger to modify it) and the *Edit Selected Logger(s)* button pressed to load the dialog shown above. At this point, any of the available Levels supported by the server can be applied to the Loggers selected and they will immediately update, as will any child Loggers without their own specific Level.

The RootLogger can be similarly edited using the button at the bottom left of the window.

3.1.1.5.6.2. ConfigurationFile Options

The ConfigurationFile Options tab allows alteration of the Level settings for the Loggers defined in the configuration file, allowing changes to persist following a restart of the server. Changes made to the configuration file are only applied automatically while the sever is running if it was configured to enable the LogWatch capability, meaning it will monitor the configuration file for changes and apply the new configuration when the change is detected. If this was not enabled, the changes will be picked up when the server is restarted. The status of the LogWatch feature is shown at the bottom of the tab. Alternatively, in the latest versions of the LoggingManagement MBean it is possible to reload the logging configuration file on demand.

Manipulating the Levels is as on the Runtime Options tab, either double-click an individual Logger entry or select multiple Loggers and use the button to load the dialog to set the new Level.



NOTE: These options modify the configuration file. Changes only take effect automatically if LogWat

A child Logger set to a non-inherited Level in the Runtime tab will retain that value after the file is rel

LoggerName
Qpid.Broker
org.apache.qpid
org.apache.qpid.server.queue.AMQQueueMBean

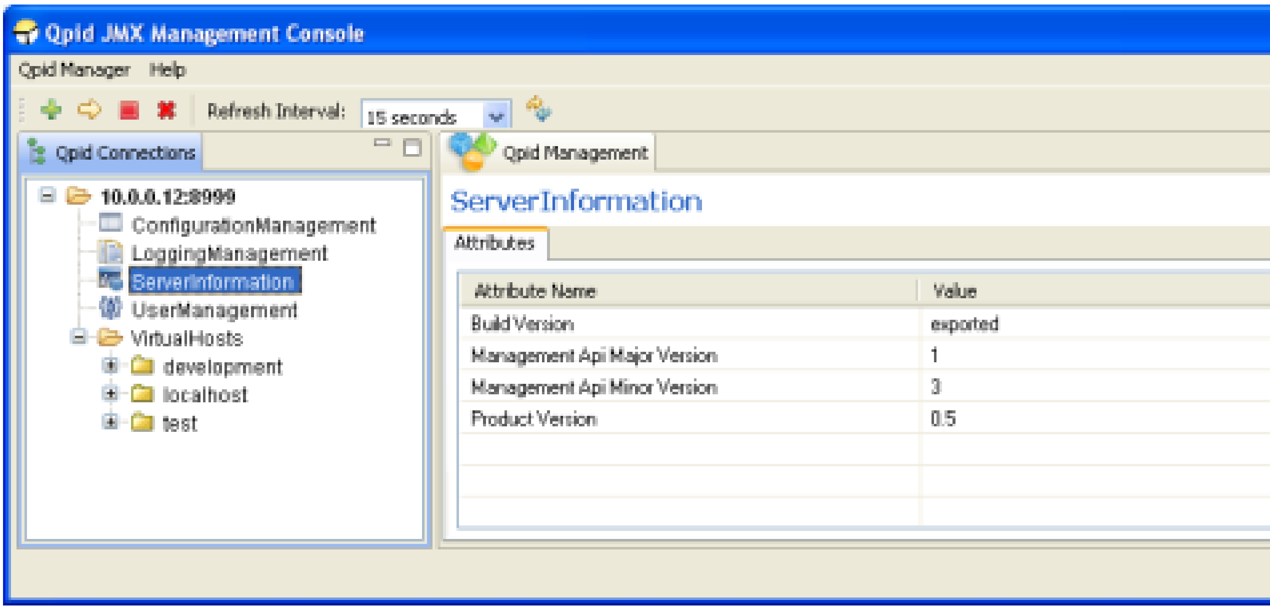
Config File RootLogger Level: INFO

LogWatch Interval: Disabled (0 sec)

One issue to note of when reloading the configuration file settings, either automatically using LogWatch or manually, is that any Logger set to a specific Level using the Runtime Options tab that is not defined in the configuration file will maintain that Level when the configuration file is reloaded. In other words, if a Logger is defined in the configuration file, then the configuration file will take precedence at reload, otherwise the Runtime options take precedence.

This situation will be immediately obvious by examining the Runtime Options tab to see the effective Level of each Logger – unless it has been altered with the RuntimeOptions or specifically set in the configuration file, a Logger Level should match that of its parent. In the latest versions of the LoggingManagement MBean, it is possible to use the RuntimeOptions to restore a child logger to inheriting from its parent by setting it with an INHERITED level that removes any previously set Level of its own.

3.1.1.5.7. ServerInformation MBean

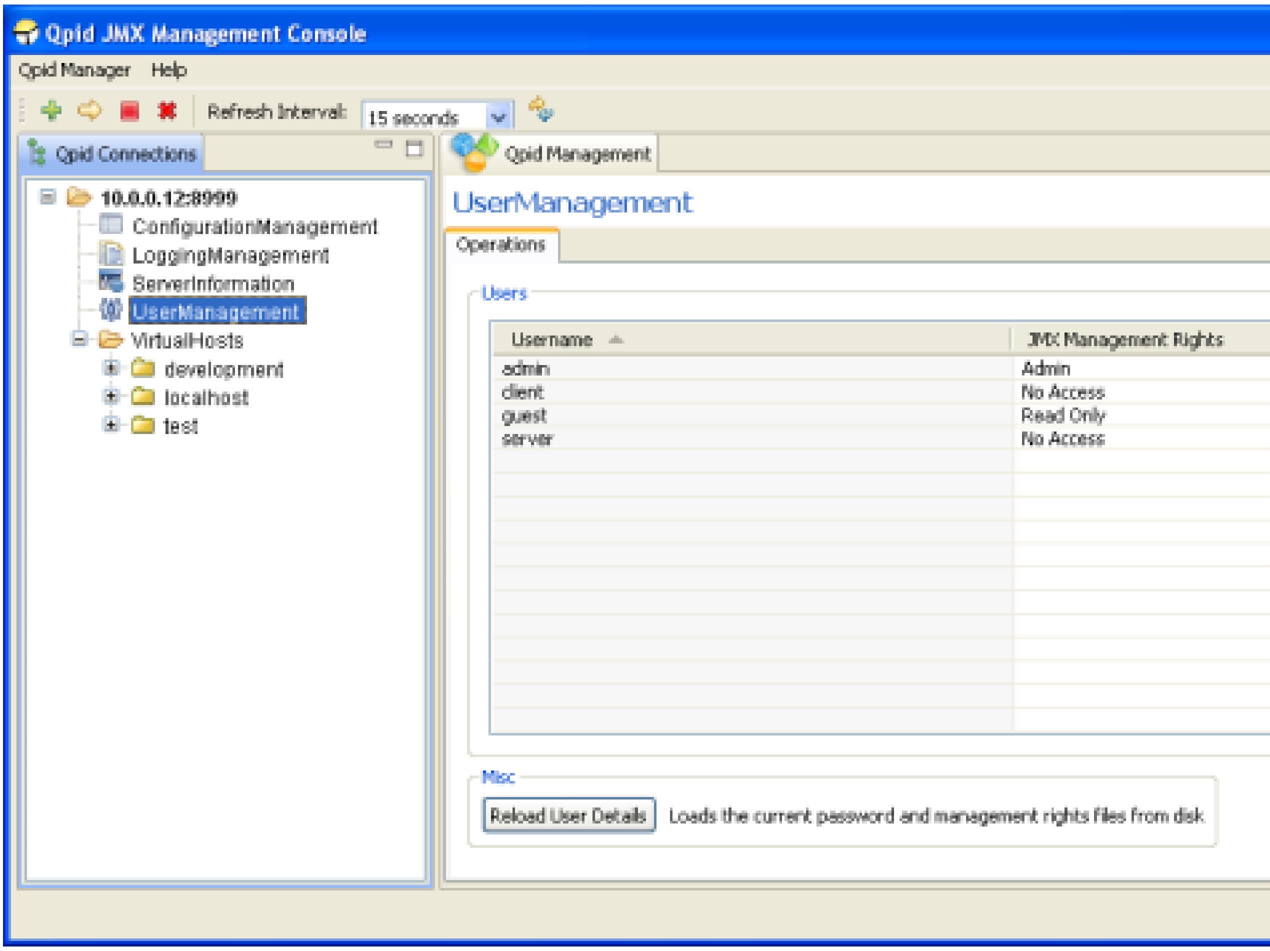


Attribute Name	Value
Build Version	exported
Management Api Major Version	1
Management Api Minor Version	3
Product Version	0.5

The ServerInformation MBean currently only conveys various pieces of version information to allow precise identification of the server version and its management capabilities. In future it is likely to convey additional server-wide details and/or functionality.

3.1.1.5.8. UserManagement MBean

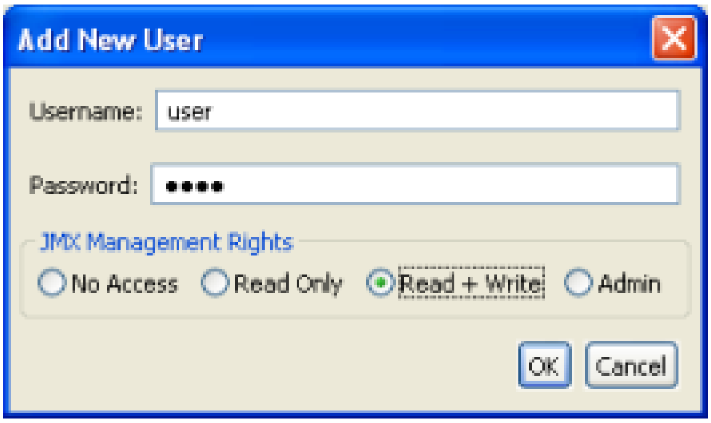
The UserManagement MBean is accessible by admin level users, and allows manipulation of existing user accounts and creation of new user accounts.



Username	JMX Management Rights
admin	Admin
client	No Access
guest	Read Only
server	No Access

Reload User Details — Loads the current password and management rights files from disk

To add a new user, press the *Add New User* button, which will load the dialog shown below.



Username: user

Password: ••••

JMX Management Rights

○ No Access ○ Read Only ● Read + Write ○ Admin

Here you may enter the new users Username, Password, and select their JMX Management Rights. This controls whether or not they have access to the management interface, and if so what capabilities are accessible. *Read Only* access allows undertaking any operations that do not alter the server state, such as viewing messages. *Read + Write* access allows use of all operations which are not deemed admin-only (such as those in the UserManagement MBean itself). *Admin* access allows a user to utilize any operation, and view the admin-only MBeans (currently these are ConfigurationManagement, LoggingManagement, and UserManagement).

One or more users at a time may be deleted by selecting them in the table and clicking the *Delete User(s)* button. The console will then prompt for confirmation before undertaking the removals.

Similarly, the access rights for one or more users may be updated by selecting them in the table and clicking the *Set Rights* button. The console will then display a dialog enabling selection of the new access level and confirmation to undertake the update.

An individual user password may be updated by selecting the user in the table in and clicking the *Set Password* button. The console will then display a dialog enabling input of the new password and confirmation to undertake the update.

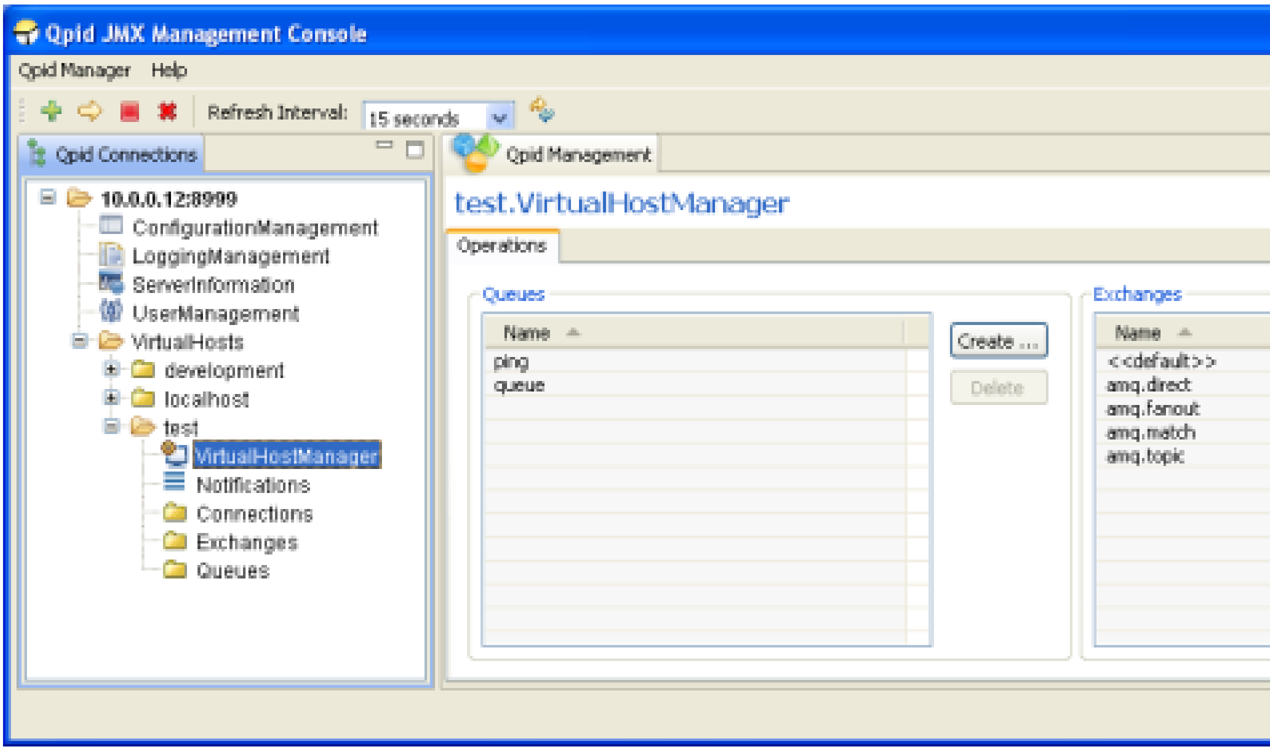
The server caches the user details in memory to aid performance. If may sometimes be necessary to externally modify the password and access right files on disk. In order for these changes to be known to the server without a restart, it must be instructed to reload the file contents. This can be done using the provided *Reload User Details* button (on older servers, only the management rights file is reloaded, on newer servers both files are. The description on screen will indicate the behaviour). After pressing this button the console will seek confirmation before proceeding.

3.1.1.5.9. VirtualHostManager MBean

Each VirtualHost in the server has an associated VirtualHostManager MBean. This allows viewing, creation, and deletion of Queues and Exchanges within the VirtualHost.

Clicking the *Create* button in the Queue section will open a dialog allowing specification of the Name, Owner (optional), and durability properties of the new Queue, and confirmation of the operation.

One or more Queues may be deleted by selecting them in the table and clicking the *Delete* button. This will unregister the Queue bindings, remove the subscriptions and delete the Queue(s). The console will prompt for confirmation before undertaking the operation.



Clicking the *Create* button in the Exchange section will open a dialog allowing specification of the Name, Type, and Durable attributes of the new Exchange, and confirmation of the operation.

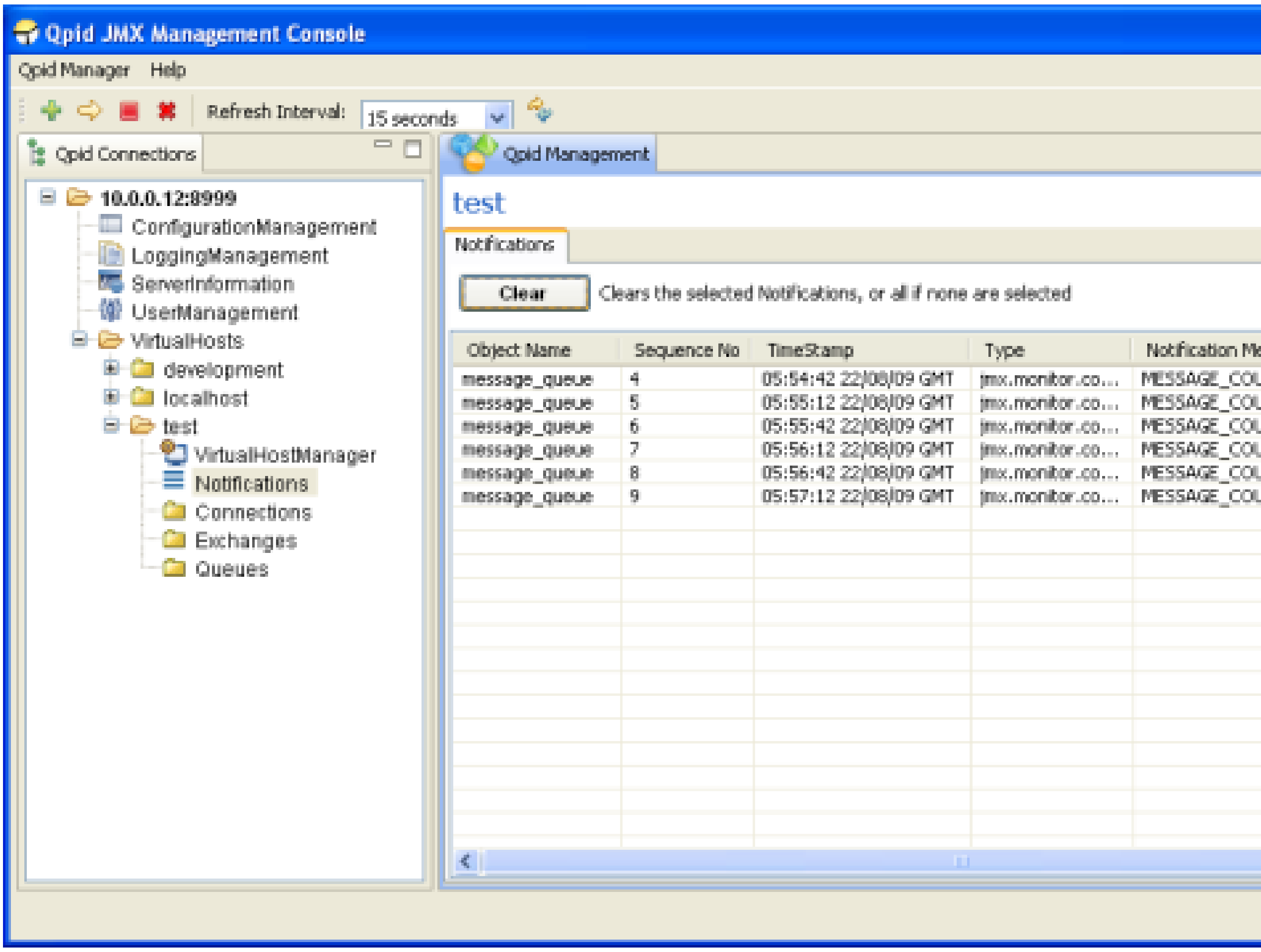
One or more Exchanges may be deleted by selecting them in the table and clicking the *Delete* button. This will unregister all the related channels and Queue bindings then delete the Exchange(s). The console will prompt for confirmation before undertaking the operation.

Double-clicking on a particular Queue or Exchange name in the tables will open the MBean representing it.

3.1.1.5.10. Notifications

MBeans on the server can potentially send Notifications that users may subscribe to. When managing an individual MBean that offers Notifications types for subscription, the console supplies a Notifications tab to allow (un)subscription to the Notifications if desired and viewing any that are received following subscription.

In order to provide quicker access to/awareness of any received Notifications, each VirtualHost area in the connection tree has a Notifications area that aggregates all received Notifications for MBeans in that VirtualHost. An example of this can be seen in the figure below.



Clears the selected Notifications, or all if none are selected

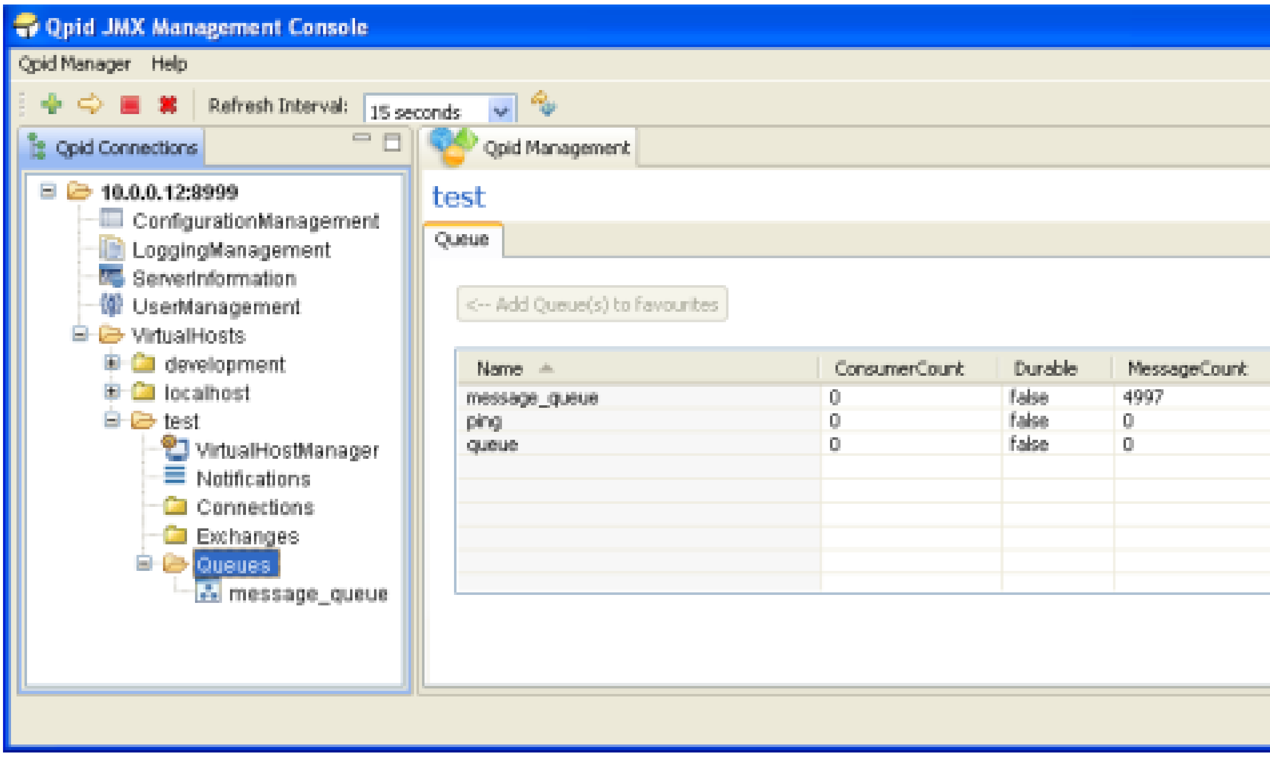
Object Name	Sequence No	TimeStamp	Type	Notification Me
message_queue	4	05:54:42 22/08/09 GMT	jmx.monitor.co...	MESSAGE_COU
message_queue	5	05:55:12 22/08/09 GMT	jmx.monitor.co...	MESSAGE_COU
message_queue	6	05:55:42 22/08/09 GMT	jmx.monitor.co...	MESSAGE_COU
message_queue	7	05:56:12 22/08/09 GMT	jmx.monitor.co...	MESSAGE_COU
message_queue	8	05:56:42 22/08/09 GMT	jmx.monitor.co...	MESSAGE_COU
message_queue	9	05:57:12 22/08/09 GMT	jmx.monitor.co...	MESSAGE_COU

All received Notifications will be displayed until such time as the user removes them, either in this aggregated view, or in the Notifications area of the individual MBean that generated the Notification.

They may be cleared selectively or all at once. To clear particular Notifications, they should be selected in the table before pressing the *Clear* button. To clear all Notifications, simply press the *Clear* button without anything selected in the table, at which point the console will request confirmation of this clear-all action.

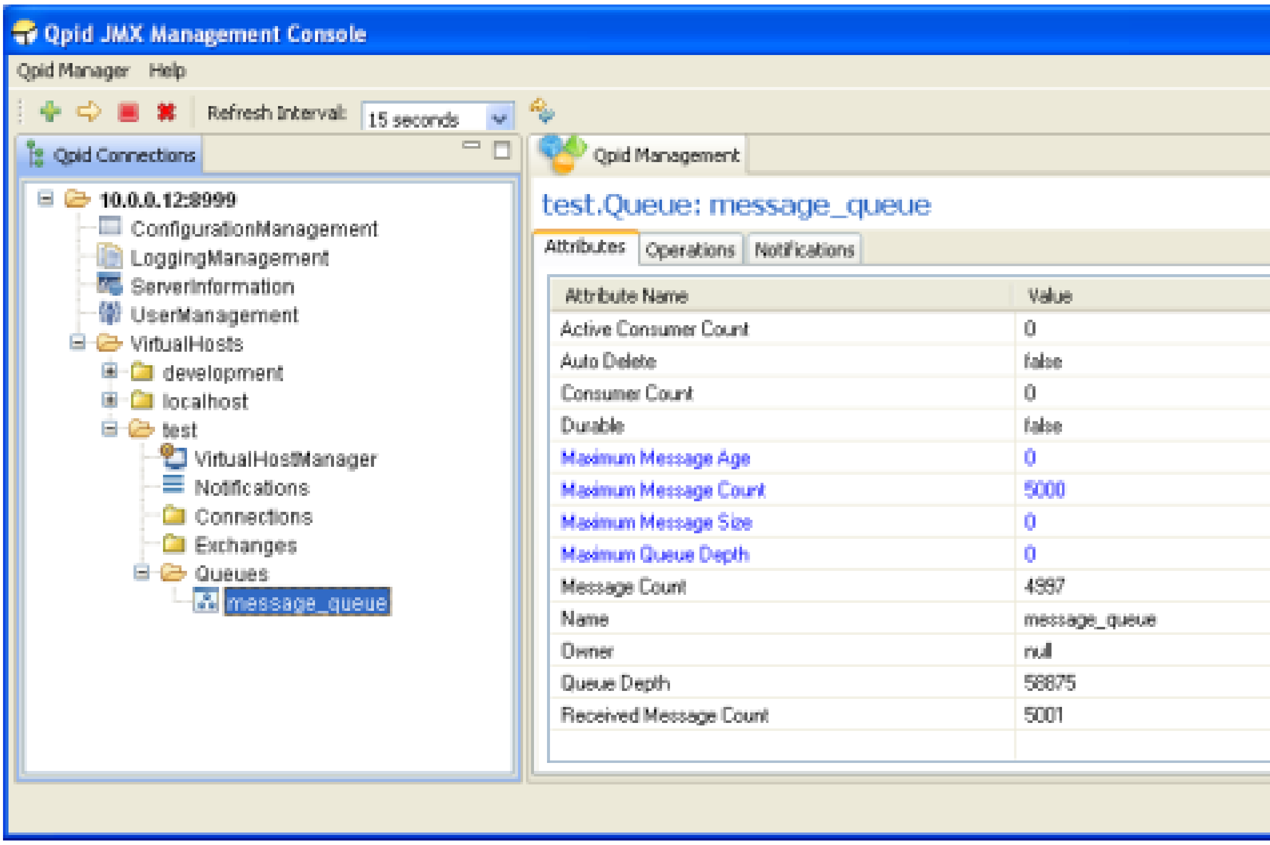
3.1.1.5.11. Managing Queues

As mentioned in earlier discussion of Navigation, Queue MBeans can be opened either by double clicking an entry in the Queues selection area, or adding a queue to the tree as a favourite and clicking on its tree node. Unique to the Queue selection screen is the ability to view additional attributes beyond just that of the Queue Name. This is helpful for determining which Queues satisfy a particular condition, e.g. having <X> messages on the queue. The example below shows the selection view with additional attributes *Consumer Count, Durable, MessageCount, and QueueDepth* (selected using the *Select Attributes* button at the bottom right corner of the table).



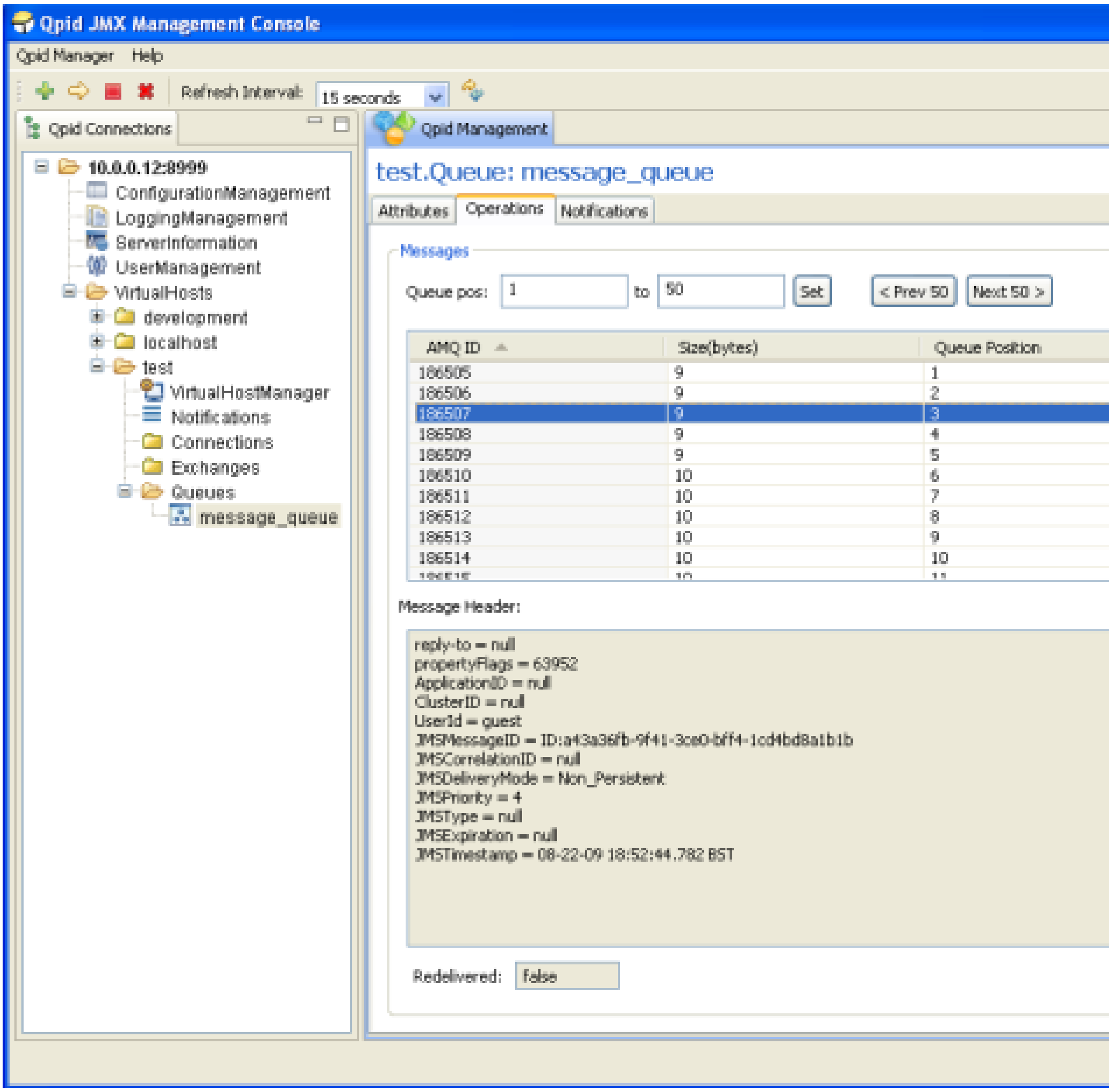
Name	ConsumerCount	Durable	MessageCount
message_queue	0	false	4997
ping	0	false	0
queue	0	false	0

Upon opening a Queue MBean, the Attributes tab is displayed, as shown below. This allows viewing the value all attributes, editing those which are writable values (highlighted in blue) if the users management permissions allow, viewing descriptions of their purpose, and graphing certain numerical attribute values as they change over time.



Attribute Name	Value
Active Consumer Count	0
Auto Delete	false
Consumer Count	0
Durable	false
Maximum Message Age	0
Maximum Message Count	5000
Maximum Message Size	0
Maximum Queue Depth	0
Message Count	4997
Name	message_queue
Owner	null
Queue Depth	58875
Received Message Count	5001

The next tab contains the operations that can be performed on the queue. The main table serves as a means of viewing the messages on the queue, and later for selecting specific messages to operate upon. It is possible to view any desired range of messages on the queue by specifying the visible range using the fields at the top and pressing the *Set* button. Next to this there are helper buttons to enable faster browsing through the messages on the queue; these allow moving forward and back by whatever number of messages is made visible by the viewing range set. The Queue Position column indicates the position of each message on the queue, but is only present when connected to newer servers as older versions cannot provide the necessary information to show this (unless only a single message position is requested).



Refresh Interval: 15 seconds

test.Queue: message_queue

Queue pos: 1 to 50

AMQ ID	Size(bytes)	Queue Position
186505	9	1
186506	9	2
186507	9	3
186508	9	4
186509	9	5
186510	10	6
186511	10	7
186512	10	8
186513	10	9
186514	10	10

Message Header:

```
reply-to = null
propertyFlags = 63952
ApplicationID = null
ClusterID = null
UserId = guest
JMSMessageID = ID:a43a36fb-9f41-3ca0-bff4-1cd4bd8a1b1b
JMSCorrelationID = null
JMSDeliveryMode = Non_Persistent
JMSPriority = 4
JMSType = null
JMSExpiration = null
JMSTimestamp = 08-22-09 18:52:44.782 BST
```

Redelivered: false

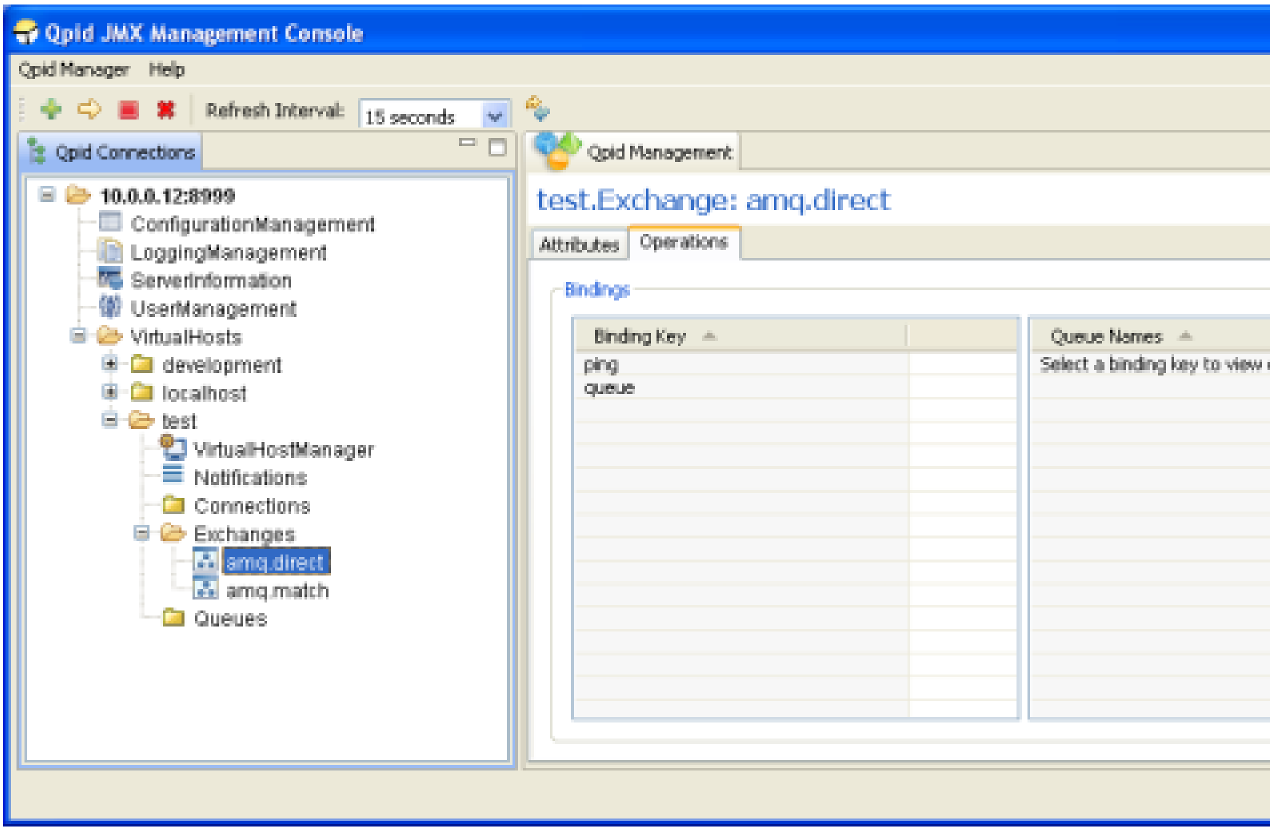
Upon selecting a message in the table, its header properties and redelivery status are updated in the area below the table. Double clicking a message in the table (or using the *View Message Content* button to its right) will open a dialog window displaying the contents of the message.

One or more messages can be selected in the table and moved to another queue in the VirtualHost by using the *Move Message(s)* button, which opens a dialog to enable selection of the destination and confirmation of the operation. Newer servers support the ability to similarly copy the selected messages to another queue in a similar fashion, or delete the selected messages from the queue after prompting for confirmation.

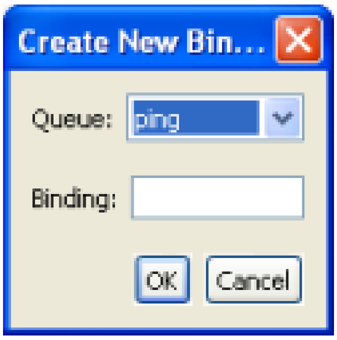
Finally, all messages (that have not been acquired by consumers) on the queue can be deleted using the *Clear Queue* button, which will generate a prompt for confirmation. On newer servers, the status bar at the lower left of the application will report the number of messages actually removed.

3.1.1.5.12. Managing Exchanges

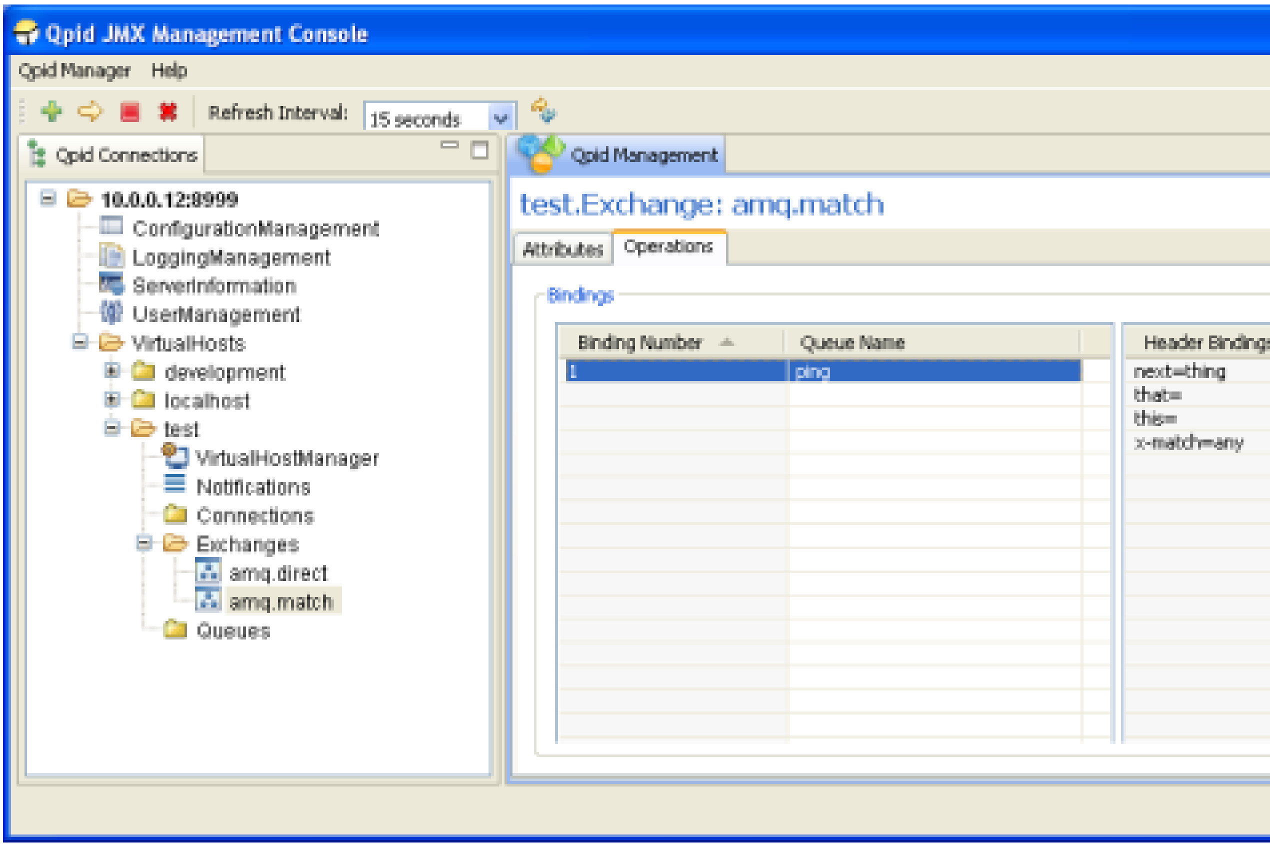
Exchange MBeans are opened for management operations in similar fashion as described for Queues, again showing an Attributes tab initially, with the Operations tab next:



Of the four default Exchange Types *(direct, fanout, headers, and topic)* all but *headers* have their bindings presented in the format shown above. The left table provides the binding/routing keys present in the exchange. Selecting one of these entries in the table prompts the right table to display all the queues associated with this key. Pressing the *Create* button opens a dialog allowing association of an existing queue with the entered Binding.



The *headers* Exchange type (default instantiation *amq.match or amq.headers*) is presented as below:



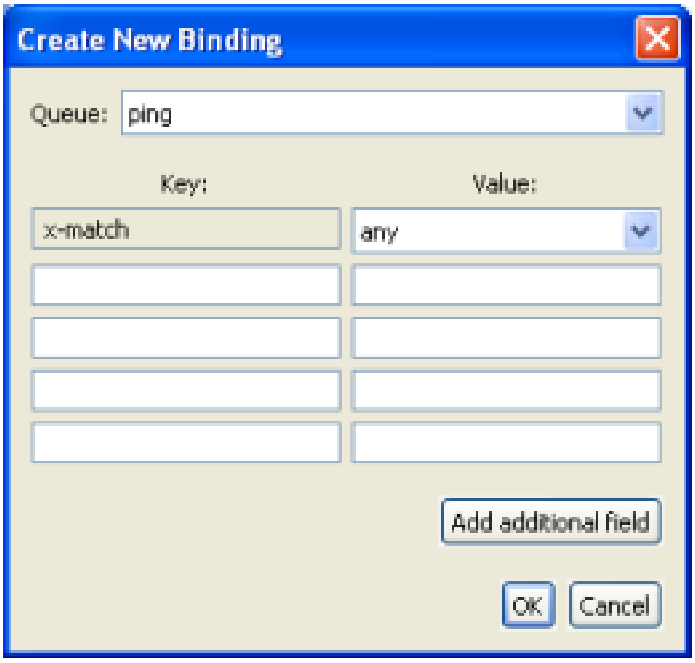
Refresh Interval: 15 seconds

test.Exchange: amq.match

Binding Number	Queue Name
1	ping

Header Binding
next=thing
that=
this=
x-match=any

In the previous figure, the left table indicates the binding number, and the Queue associated with the binding. Selecting one of these entries in the table prompts the right table to display the header values that control when the binding matches an incoming message.



Queue: ping

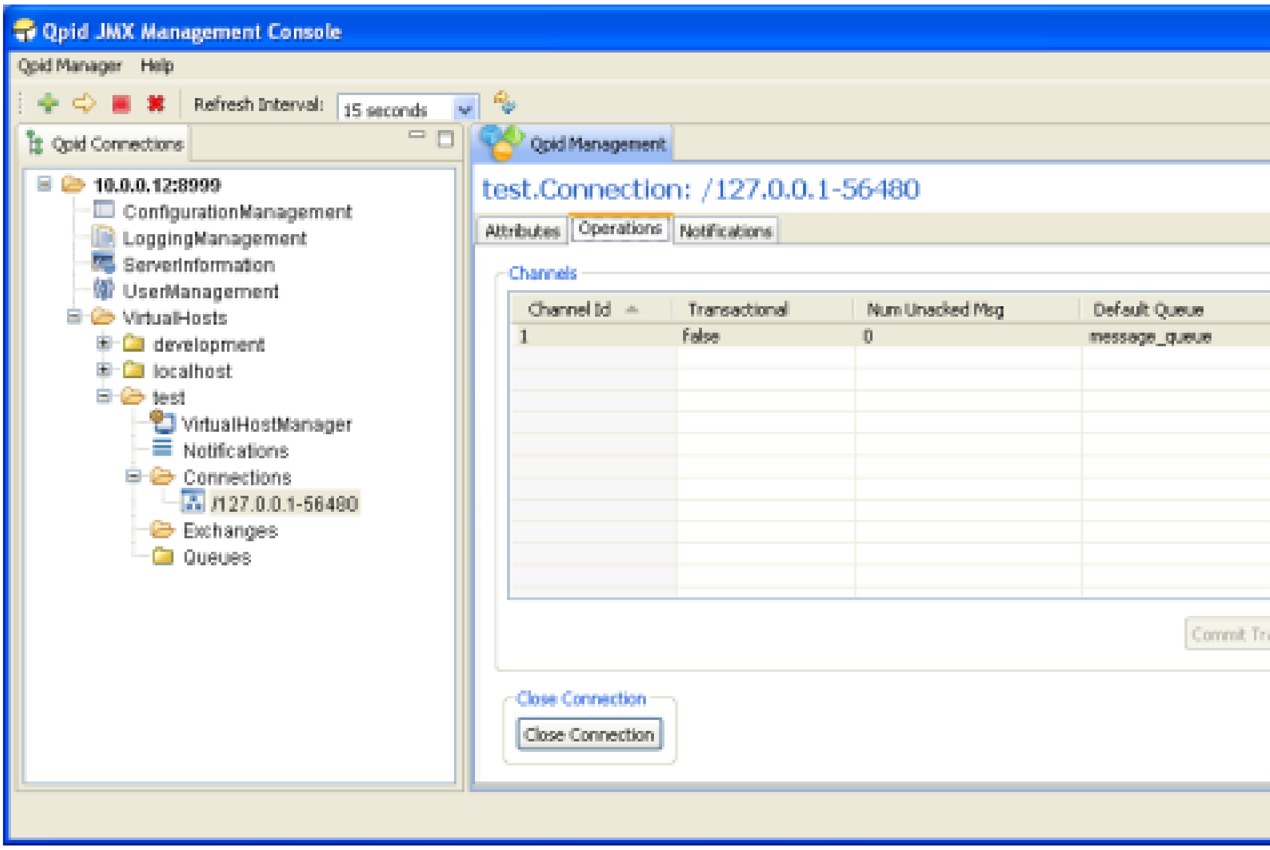
Key:	Value:
x-match	any

Pressing the *Create* button when managing a *headers* Exchange opens a dialog allowing creation of a new binding, associating an existing Queue with a particular set of header keys and values. The *x-match* key is required, and instructs the server whether to match the binding with incoming messages based on ANY or ALL of the further key-value pairs entered. If it is desired to enter more than 4 pairs, you may press the *Add additional field* button to create a new row as many times as is required. When managing a *headers* Exchange, double clicking an entry in the left-hand table will open the MBean for the Queue specified in the binding properties.

When managing another Exchange Type, double clicking the Queue Name in the right-hand table will open the MBean of the Queue specified.

3.1.1.5.13. Managing Connections

Exchange MBeans are opened for management operations in similar fashion as described for Queues, again showing an Attributes tab initially, with the Operations tab next, and finally a Notifications tab allowing subscription and viewing of Notifications. The Operations tab can be seen in the figure below.



Refresh Interval: 15 seconds

test.Connection: /127.0.0.1-56480

Channel Id	Transactional	Num Unacked Msg	Default Queue
1	false	0	message_queue

The main table shows the properties of all the Channels that are present on the Connection, including whether they are *Transactional*, the *Number of Unacked Messages* on them, and the *Default Queue* if there is one (or *null* if there is not).

The main operations supported on a connection are Commiting and Rolling Back of Transactions on a particular Channel, if the Channel is Transactional. This can be done by selecting a particular Channel in the table and pressing the *Commit Transactions* or *Rollback Transactions* buttons at the lower right corner of the table, at which point the console will prompt for confirmation of the action. These buttons are only active when the selected Channel in the table is Transactional.

The final operation supported is closing the Connection. After pressing the *Close Connection* button, the console will prompt for confirmation of the action. If this is carried out, the MBean for the Connection being managed will be removed from the server. The console will be notified of this by the server and display an information dialog to that effect, as it would if any other MBean were to be unregistered whilst being viewed.

Double clicking a row in the table will open the MBean of the associated *Default Queue* if there is one.

3.1.1.6. Qpid Management Features

Management tool: See our ??? for details of how to use various console options with the Qpid management features.

The management of QPID is categorised into following types-

1. Exchange
2. Queue
3. Connection
4. Broker

1) Managing and Monitoring Exchanges: Following is the list of features, which we can have available for managing and monitoring an Exchange running on a Qpid Server Domain-

1. Displaying the following information for monitoring purpose-
    a. The list of queues bound to the exchange along with the routing keys.
    b. General Exchange properties(like name, durable etc).
2. Binding an existing queue with the exchange.

2) Managing and Monitoring Queues: Following are the features, which we can have for a Queue on a Qpid Server Domain-

1. Displaying the following information about the queue for monitoring purpose-
    a. General Queue properties(like name, durable, etc.)
    b. The maximum size of a message that can be accepted from the message producer.
    c. The number of the active consumers accessing the Queue.
    d. The total number of consumers (Active and Suspended).
    e. The number of undelivered messages in the Queue.
    f. The total number of messages received on the Queue since startup.
    g. The maximum number of bytes for the Queue that can be stored on the Server.
    h. The maximum number of messages for the Queue that can be stored on the Server.
2. Viewing the messages on the Queue.
3. Deleting message from top of the Queue.
4. Clearing the Queue.
5. Browsing the DeadMessageQueue - Messages which are expired or undelivered because of some reason are routed to the DeadMessageQueue. This queue can not be deleted. [Note: The is open because it depends on how these kind of messages will be handeled?]

3) Managing and Monitoring Connections: Following are the features, which we can have for a connection on a QPID Server Domain-

1. Displaying general connection properties(like remote address, etc.).
2. Setting maximum number of channels allowed for a connection.

3. View all related channels and channel properties.
4. Closing a channel.
5. Commit or Rollback transactions of a channel, if the channel is transactional.
6. Notification for exceeding the maximum number of channels.
7. Dropping a connection.
8. The work for ??? implies that there are potentially some additional requirements
    a. Alert when tcp flow control kicks in
    b. Information available about current memory usage available through JMX interface
    c. Dynamic removal of buffer bounds? (fundamentally not possible with TransportIO)
    d. Management functionality added to JMX interface - UI changes?

4) Managing the Broker: Features for the Broker-

1. Creating an Exchange.
2. Unregistering an Exchange.
3. Creating a Queue.
4. Deleting a Queue.

Chapter 4. Management Tools

4.1. Qpid Java Broker Management CLI

4.1.1. How to build Apache Qpid CLI

4.1.1.1. Build Instructions - General

At the very beginning please build Apache Qpid by refering this installation guide from here ???.

After successfully build Apache Qpid you'll be able to start Apache Qpid Java broker,then only you are in a position to use Qpid CLI.

4.1.1.2. Check out the Source

First check out the source from subversion repository. Please visit the following link for more information about different versions of Qpid CLI.

???

4.1.1.3. 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 !

4.1.1.4. Building Apache Qpid CLI

This project is currently having only an ant build system.Please install ant build system before trying to install Qpid CLI.

4.1.1.5. Compiling

To compile the source please run following command

```
ant compile
```

To compile the test source run the following command

```
ant compile-tests
```

4.1.1.6. Running CLI

After successful compilation set QPID_CLI environment variable to the main source directory.(set the environment variable to the directory where ant build script stored in the SVN checkout).Please check whether the Qpid Java broker is up an running in the appropriate location and run the following command to start the Qpid CLI by running the qpid-cli script in the bin directory.

$QPID_CLI/bin/qpid-cli -h <hostname of the broker> -p <broker running port> For more details please have a look in to README file which ships with source package of Qpid CLI.

4.1.1.7. Other ant targets

For now we are supporting those ant targets.

ant clean	Clean the complete build including CLI build and test build.
ant jar	Create the jar file for the project without test cases.
ant init	Create the directory structure for build.
ant compile-tests	This compiles all the test source.
ant test	Run all the test cases.